Exoplanet-Ready Computing: Building and Living With a Truly Offline Machine in a Network-Addicted Era

Douglas C. Youvan

doug@youvan.com

www.youvan.ai

September 24, 2025

Fifty years ago a PDP-8 at Berkeley worked just fine without ever touching a network. Today, almost every assumption in mainstream computing presumes you are online—licenses, updates, drivers, identity, help, even basic setup flows. This essay documents the construction and daily use of a fully offline "proof device" on a Ryzen-7 mini-PC, air-gapped from the modern web and supplied only by a sneaker-net USB shuttle. The goal isn't nostalgia; it's resilience and sovereignty. What actually breaks when you sever the wire? What still works beautifully? Where are vendors selling convenience versus enforcing dependence? From out-of-box experiences that resist local accounts, to toolchains that insist on remote registries, to hidden time/DRM assumptions, we map the frictions and the remedies. We also show why an exoplanet-ready computer is not a stunt: offline brings determinism, auditability, and security properties that the connected world can't. Along the way we design a safe "USB shuttle protocol," curate local mirrors for packages and docs, and lay out a rubric—Exoplanet Readiness Levels—for any workstation. The punchline: once you feel the quiet of a machine that owes nothing to the network, you see dependence everywhere—and you can choose where to keep it, and where to cut it.

Keywords: air-gap, offline-first, reproducibility, USB sneaker-net, local mirrors, OOBE, determinism, supply-chain, Beelink Ryzen 7, exoplanet readiness.

A collaboration with GPT-5. CC4.0. 82 pages.

Outline

1. The experiment and the memory
   1.1 PDP-8 vs 2025: what “offline” meant then and now
   1.2 The proof device: scope, constraints, success criteria
2. Where modern computing quietly assumes a network
   2.1 Identity and OOBE frictions (local users, licensing expectations)
   2.2 Drivers/firmware and time: the hidden dependence on NTP and portals
   2.3 Toolchains: package registries, SDK telemetry, doc hosting
   2.4 “Help” as a design crutch: URLs instead of manuals
3. Sales pitch or essential? A dependency triage
   3.1 Comforts (nice-to-have): cloud sync, live copilot, push docs
   3.2 Essentials (actually required): activation, updates, revocation lists
   3.3 Ethics: vendor paternalism vs user sovereignty
4. Designing an exoplanet-ready workstation
   4.1 Hardware choices and BIOS/UEFI settings that favor autonomy
   4.2 Filesystems, partitioning, and backup images that restore offline
   4.3 Deterministic builds: pinning compilers, libs, and flags
   4.4 Local time discipline: manual NTP stand-in, monotonic time notes
5. The USB shuttle protocol (safe sneaker-net)
   5.1 Roles: ferry, quarantine, staging, release
   5.2 Hash and signature routine (sha256 + detached sig)
   5.3 Cold-scanning and content policies (no macros, no autorun)
   5.4 Chain-of-custody log you can actually keep
6. Packaging an offline universe
   6.1 Mirroring core docs: language specs, manpages, PDFs
   6.2 Mirroring package repos and wheels; freezing lockfiles
   6.3 Building local indices (search without the web)
   6.4 Teaching the toolchain to prefer “file://” over “https://”
7. Operating without telemetry
   7.1 What you lose: autocomplete from the hive mind
   7.2 What you gain: reproducibility, performance determinism, privacy
   7.3 Benchmarks: compile time, proof time, variance across runs

8. Exoplanet Readiness Levels (ERL)
   ERL-0: boots offline
   ERL-1: installs + updates from local media
   ERL-2: full toolchain mirrors and docs local
   ERL-3: deterministic builds + reproducible proofs
   ERL-4: recover from bare metal with no network
   ERL-5: multi-year sustainment (keys, drivers, firmware, time)

9. Case study: the Beelink proof device
   9.1 OOBE to first proof: gotchas and workarounds (high-level)
   9.2 Driver + firmware strategy without calling home
   9.3 The daily loop: ingest → verify → stage → run → attest

10. What this changes in you
    10.1 Attention without pings: flow as a system feature
    10.2 Sovereign failure modes (and why they're kinder)
    10.3 Choosing dependence deliberately

11. Policy and pedagogy
    11.1 "Offline-first" as a right, not a curiosity
    11.2 Vendor minimums: local account, local docs, local install paths
    11.3 Teaching computing that still works on an exoplanet

12. Conclusion
    The quiet machine and the loud world: where to reconnect, and where not to

Sidebar A — Air-gap Ops Playbook (10 rules)

1. Two USBs: ferry and quarantine—never reuse across roles.
2. Hash everything: sha256sum >LOG; verify before mount/use.
3. Sign releases: detached signatures with a long-lived offline key.
4. No autorun, ever; open via hash-listed paths only.
5. Convert docs to stable, offline formats (PDF/CHM/epub).
6. Mirror once, use for months: versioned snapshots beat "latest".
7. Log ingress/egress in plain text with timestamps.
8. Prefer static builds; avoid dynamic plugins.
9. Keep a golden bare-metal image and test restores quarterly.

10. Treat time as data: record local time policy in every run log.

Sidebar B — OOBE & vendor lock-ins (high-level, ethical)

- Prefer official/local-account setup modes and documented deployment tools (e.g., offline provisioning with answer files or device images).
- Avoid gray-area hacks that might violate terms or impair security features.
- Keep a clean written record: what steps required the network and why you declined them.

Appendix — Minimal offline kit (practical checklist)

- BIOS/UEFI update on a throwaway USB; verify vendor checksums offline.
- OS install media + cumulative updates on USB; scripted post-install.
- Language toolchains, compilers, and package mirrors pinned to exact versions.
- Local doc set (language refs, manpages, PDFs, API docs).
- Hash/sign tools (sha256, minisign or gpg), and a printed fingerprint.
- A4 binder: runbooks, recovery steps, and the ERL rubric.

1) The experiment and the memory

1.1 PDP-8 vs 2025: what “offline” meant then and now

Then (PDP-8, c. 1970s):

- **Provisioning:** Media were self-sufficient (paper tape, DECtape, mag tape). No implicit license pings, no activation servers.
- **Identity:** Single-machine accounts; no cloud ID. Admin == physical access.
- **Documentation:** Manuals shipped with the machine. Knowledge lived in binders and printouts.
- **Time:** Wall clock was approximate; software rarely depended on trusted time. No CRL/OCSP, no TLS.
- **Toolchain:** Compilers/interpreters were part of the local distribution. “Install” meant copy/load.
- **Failure modes:** Mechanical wear, media errors. Debug path was local and transparent.
- **Security surface:** Physical custody. “Air-gap” was the default.

Now (Ryzen-7 mini-PC, 2025):

- **Provisioning:** OS installers, drivers, and firmware designed around an assumed network path; OOBE nudges toward online accounts.
- **Identity:** Cloud identity by default; local accounts are possible but discouraged.
- **Documentation:** URLs instead of manuals; help buttons open web pages.
- **Time:** Trusted time is entangled with security (code signing, cert validity, logs). Many tools assume NTP.
- **Toolchain:** Package managers default to remote registries (pip, cargo, npm). “Install” means “resolve online.”
- **Failure modes:** Network expectations masquerade as “bugs” when offline (timeouts, broken UX paths).
- **Security surface:** Supply chain and telemetry by default; true air-gap is now a conscious design.

Implications for "offline":

- **1970s offline** = normal operation.
- **2025 offline** = counter-cultural mode that must be engineered: pre-mirrors, deterministic builds, local docs, time discipline, and an explicit ferry (USB shuttle) with hygiene.

Why this matters: The modern stack has quietly made liveness on the network a prerequisite for convenience—and sometimes for basic functionality. Reversing that requires a deliberate architecture, not nostalgia.

1.2 The proof device: scope, constraints, success criteria

Scope (what this box is and is not):

- **Is:** A self-contained mathematics/proof workstation that can:
    1. boot,
    2. build its toolchain,
    3. run proof search/checking jobs, and
    4. produce attestable artifacts,
       all **without** a network interface.
- **Is not:** A general desktop for browsing/communications; no background updates; no cloud sync; no copilot.

Hardware/OS baseline (concrete):

- **Platform:** Beelink mini-PC, Ryzen 7 (8C/16T), 32 GB RAM, 1–2 TB SSD.
- **Firmware:** UEFI with Secure Boot configurable; TPM present but not required for operation.
- **OS:** Windows 11 Pro installed and serviced entirely from removable media (or an offline-prepared image).
- **I/O policy:** Network adapters disabled; only human I/O (keyboard, display) and a **two-stick** USB shuttle workflow.

Operational constraints (hard rules):

- **C0. Air-gap:** No wired/wireless networking. Adapters disabled in firmware and OS.
- **C1. Provenance:** Every ingress via USB passes quarantine → hash → sign-verify before entering the working set.
- **C2. Determinism:** Builds and runs are pinned by versioned manifests; randomness is seeded and logged.
- **C3. Time discipline:** A local, declared time policy: set once per session; monotonic timing recorded; no external NTP.
- **C4. Docs local:** All help/specs man-page/PDF/CHM/EPUB mirrored locally; help links resolve to file:// paths.
- **C5. Rebuildability:** From bare metal + offline kit, the box can be rebuilt to a bit-identical working state.
- **C6. Audit trail:** Plain-text logs for ingress/egress, builds, proofs, and signatures with stable filenames and hashes.

Success criteria (measurable):

- **S1. Reproducible install:** A scripted rebuild from bare metal reaches “ready” state with 0 network calls and completes within T_install_max hours. Evidence: install log + final system hash inventory.
- **S2. Deterministic toolchain:** Given a lockfile set L, toolchain_build(L) produces identical binaries across 3 runs:
  hash(bin_run1) == hash(bin_run2) == hash(bin_run3).
  Variance in build wall-clock <= 5 % under identical thermal conditions.
- **S3. Deterministic proofs:** For a fixed problem set P and seed s, the proof runner yields identical accepted artifacts and hashes across N >= 5 runs:
  outputs_run_k ::= {artifact_i, sha256_i}.
  All runs produce the same index and same sha256 list order.
- **S4. No network dependency:** System-wide network monitoring shows zero attempted outbound connections during install, build, or proof runs. Evidence: Windows firewall logs + disabled NICs + offline capture of relevant ETW providers.
- **S5. Attestable artifacts:** Every build/proof artifact A ships with:
  sha256(A), size(A), utc_local_timestamp, and a detached signature sig(A) created by an offline key. A manifest file indexes all artifacts.

- **S6. Recovery:** Quarterly bare-metal restore test succeeds: boot → restore image → validate hashes of key directories (toolchain, proofs, docs) match canonical manifest. Time to recover <= T_recover_max.
- **S7. Sustainment:** All keys, installers, drivers, manifests, and docs are present in the offline kit to operate for Y years without the network (e.g., Y = 3). Evidence: inventory.txt with counts, versions, and expiry dates where applicable.

Process primitives (copy-safe notation):

- **Hash:** sha256(file) -> hex64.
- **Sign:** sign_detached(file, private_key_offline) -> .sig.
- **Verify:** verify(file, .sig, public_key) -> {ok|fail}.
- **Run envelope:**
  run_id := yyyymmdd_HHMM_local;
  seed := fixed uint64;
  env := {PATH, PYTHONHASHSEED=seed, TZ=local_declared}.
  For each job j in plan:
    1. log(start, run_id, j),
    2. execute(j, env),
    3. log(stop, status, sha256(outputs_j)).

Minimal directory layout (stable, printable):

```
C:\proofbox\
  00_docs\            # local manuals/specs
  01_kits\            # installers, drivers, firmware (immutable)
  02_toolchain\
    locks\            # lockfiles, manifests
    builds\           # built bins with hashes
  03_inputs\          # curated math instances (read-only once staged)
  04_runs\
    2025-09-24_2000\  # run_id
```

logs\

outputs\

manifest.txt

90_quarantine\ # raw ferry USB drops; no execution here

91_stage\ # verified content awaiting release

keys\ # offline public keys (private lives offbox)

bin\ # sha256sum.exe, minisign.exe, etc.

Roles and human loop (who does what on an air-gap):

- **Ferry operator (online box):** Download → hash → print checksums → move to ferry USB.
- **Quarantine officer (offline box):** Copy to 90_quarantine → verify hashes/signatures → promote to 91_stage or reject.
- **Release manager:** Move from 91_stage to 01_kits / 02_toolchain / 03_inputs with an appended entry in ingress.log.
- **Runner:** Execute builds/proofs; produce manifests and signatures; append to egress.log for any artifacts to be exported.

Initial acceptance test (day-1, practical):

1. Disable NICs in BIOS and OS. Confirm no interfaces up.
2. Install OS from USB; create local admin.
3. Apply offline cumulative updates from 01_kits.
4. Install Python and tools from 01_kits; set PATH.
5. Verify docs open from 00_docs without network.
6. Build a tiny deterministic sample (hello toolchain): compile → hash → repeat 3x.
7. Run a toy proof suite P0 with seed 12345, produce manifest and signatures.
8. Confirm zero outbound network attempts in firewall/ETW logs.
9. Snapshot a “golden image” and record its sha256 inventory.

This frames the experiment in terms that are historically honest (PDP-8) and operationally concrete (your Beelink proof device), with pass/fail gates you can run quarterly. When you're ready, we can move to Section 2 (network assumptions) or flesh out your Section 5 USB shuttle SOP with exact commands for hashing, signing, logging, and promotion.

2) Where modern computing quietly assumes a network

2.1 Identity and OOBE frictions (local users, licensing expectations)

What the stack assumes

- You will authenticate with a cloud ID during Out-of-Box Experience (OOBE).
- You will accept telemetry defaults and "connect to finish setup."
- License activation will occur online in the background.

Offline-first stance

- Prefer editions that officially support local accounts and offline provisioning (e.g., Windows Pro/Education, mainstream Linux distros, *not* cloud-managed builds).
- Treat the whole OOBE as a deployment step you control, not a UI you must satisfy.

Clean, supportable paths

- **Local account:** Use the documented "local account / domain join" path offered in Pro editions. If your UI wording changes across builds, stop and use a prebuilt image instead of chasing dark-pattern prompts.
- **Provisioning vs clicking:** Create your own install media or a reference image that already has a local admin, policies, and packages. On Windows this means: install once → generalize with sysprep → capture → deploy from USB. On Linux: preseed/kickstart or build a remastered ISO.
- **Licensing:** Plan licensing up front. Some products require periodic online revalidation; avoid those for exoplanet readiness. Where offline activation is supported, use official phone or offline activation flows and keep proofs (dates, codes) in your binder.

Do / Don't

- DO keep a written "OOBE transcript" (every screen you accept/decline).
- DO export a post-install policy baseline (local users, audit, firewall).

- DON’T depend on one-time UI tricks that may break next release.
- DON’T log in with any personal cloud ID “just to get past this screen.”

ERL check

- ERL-1 if: you can reach a local desktop with a local admin and zero network calls, reproducibly, from your USB installer or captured image.

2.2 Drivers/firmware and time: the hidden dependence on NTP and portals

What the stack assumes

- Driver updates stream via Windows Update or distro repos.
- Firmware is delivered by vendor portals or live installers.
- System time is “correct” via NTP, so certs, code-signing, and logs align.

Offline-first stance

- Curate drivers and firmware as **versioned, hash-verified bundles** on your kit USB. Prefer vendors that publish standalone updaters with checksums.
- Declare a **time policy**: how the system clock is set, verified, and recorded without NTP.

Practical moves

- **Drivers:** Download WHQL driver packs from vendor sites and/or Windows Update Catalog on the online box. Keep per-device folders: vendor_model_version/ with driver.inf, readme.txt, sha256sums.txt. Install with Device Manager “Have Disk…” or pnputil. Freeze versions once validated.
- **Firmware (UEFI/SSD):** Prefer capsule/USB updaters that run pre-boot. Store copies with vendor checksums. Never run firmware tools that *require* a live portal.
- **Time discipline:**
    - Pick a reference (e.g., an accurate wristwatch or a single “time master” box on your bench that you keep disciplined by hand).
    - On first boot each session, set local time once; record: time_policy.txt with who/when/how.
    - Avoid back-dating after you’ve produced signed artifacts—monotonicity matters.

- Be aware: expired code-signing timestamps or CRL/OCSP checks can cause warnings. Your offline rule: accept only artifacts you already verified and stored; do not introduce new signed binaries that you cannot validate offline.

Do / Don't

- DO keep a printed table of device hardware IDs → driver package versions.
- DO confirm device manager has zero "unknown devices" before snapshotting your golden image.
- DON'T allow automatic driver search; it will try to go online.
- DON'T rely on scheduled time sync tasks; disable them.

ERL check

- ERL-2 if: all drivers and the last validated firmware can be installed from your kit, and the machine's time policy is written, applied, and logged with no NTP.

2.3 Toolchains: package registries, SDK telemetry, doc hosting

What the stack assumes

- Package managers (pip, cargo, npm, NuGet) resolve against public registries.
- SDKs ping home (analytics, templates, workloads).
- Official docs live behind https:// links and update continuously.

Offline-first stance

- Mirror once, pin forever. Build a **frozen toolchain** and tell tools to prefer file:// over https://.
- Turn off analytics by config, not by hope.

Python example (works well on Windows/Linux)

- On the online box:
  - python -m pip download -r locks\requirements.txt -d wheels\
  - Save exact versions into locks\requirements.txt.

- On the offline box:
    - pip install --no-index --find-links wheels\ -r locks\requirements.txt
- Bake PIP_NO_INDEX=1 and PIP_FIND_LINKS=file:///C:/proofbox/02_toolchain/wheels into user env.

Rust (cargo)

- Use a **local registry mirror** or a sparse index snapshot.
    - cargo vendor to create vendor/ and add to .cargo/config:
    - [source.crates-io]
    - replace-with = "vendored-sources"
    - [source.vendored-sources]
    - directory = "vendor"
- Pin toolchain with rustup offline tarballs; store them in your kit and install with rustup-init.exe offline args or plain unpacked toolchains.

Node.js (npm/pnpm)

- Prefer **pnpm** with an offline store, or build an **npm-offline-cache** directory with exact tarballs.
- Commit package-lock.json or pnpm-lock.yaml into locks/.
- Configure .npmrc or .pnpmfile.cjs to use file: tarballs and disable audit/telemetry.

.NET / C++

- Use **offline installers** (Visual Studio Build Tools, .NET SDK) and **Layout** features to create a local cache.
- For C/C++ libs, vendor the source or prebuilt static libs with exact versions, avoid “fetch at build” scripts.

Telemetry and “phone home”

- Set environment/config flags to opt out: e.g., PYTHONNOUSERSITE=1, DOTNET_CLI_TELEMETRY_OPTOUT=1, CARGO_TERM_COLOR=never (cosmetic), disable template/download features.

- Validate with a test: build the same project while NICs are disabled and firewall logs are enabled—zero outbound attempts.

Docs

- Mirror language specs, API references, and manpages as PDFs/CHM/EPUB.
- Generate local docs where possible: pydoc -w, Sphinx HTML, Rust cargo doc --offline, Doxygen for C/C++.
- Build a local search index (DocSearch clone or simple static index) and bookmark file:///C:/proofbox/00_docs/index.html.

Do / Don't

- DO freeze exact versions in lockfiles; store alongside wheels/tarballs.
- DO test a clean rebuild with the network device physically absent.
- DON'T allow "latest" tags or floating version ranges in a proof box.
- DON'T accept SDKs that *require* periodic online validation.

ERL check

- ERL-3 if: you can build your full toolchain and your proof targets from a cold machine using only your 02_toolchain/ mirrors and locks/, with NICs disabled.

2.4 "Help" as a design crutch: URLs instead of manuals

What the stack assumes

- When you click Help, it opens a website.
- Error messages include "Learn more" links that require connectivity.
- Vendors publish "living docs" without stable offline snapshots.

Offline-first stance

- Replace "Help → URL" with "Help → file://local_manual". Build your own shelf of stable references and wire apps to it where possible.

Practical moves

- **Local help portal:** Create 00_docs/index.html as your offline start page with sections: OS, shell, compilers, libs, proofs, SOPs, and your own Runbooks. Add a search box powered by a static index.
- **Capture core docs:** Save PDF/EPUB versions of official docs at the versions you actually use. Keep a docs_manifest.txt with titles, versions, sha256.
- **Error dictionary:** Grow a lightweight "error lexicon" in your binder. For each recurring error, keep: error text, root cause, fix, and offline reference page number or local URL.
- **In-app bindings:** Where tools allow help.url or doc path configuration, point them to file:///C:/proofbox/00_docs/.... For CLI tools, append a --help-local alias in your wrapper scripts that opens the right local page.

Do / Don't

- DO assume links will break; PDFs will not. Keep both HTML and PDF when possible.
- DO print a thin "survival" booklet (table of contents to your local docs, common commands, and SOPs).
- DON'T rely on community wikis for critical steps; snapshot them as PDFs if you must use them.

ERL check

- ERL-2 if: every tool you depend on has at least one local reference (spec or manual) reachable via file://.
- ERL-3 if: your own SOPs and error lexicon live offline and are versioned with your builds.

One quick validation drill for Section 2

- Power down, unplug any network, disable NICs in BIOS.
- Fresh boot.
- Attempt: install a driver from your kit, build your smallest proof target, open help for Python and your proof runner, read a "how-to" page, and set the system clock forward by 6 months.
- Pass if: all five steps succeed locally, logs show zero outbound attempts, and no unexplained warnings appear due to time shift.

3) Sales pitch or essential? A dependency triage

3.1 Comforts (nice-to-have): cloud sync, live copilot, push docs

Definition: Improves convenience or speed when online, but **does not expand the offline proof box's capability set**. Losing it should not block install, build, or proof.

Common comforts

- **Cloud sync & backups:** OneDrive/Drive/iCloud, roaming profiles, settings sync.
- **Live copilots & hosted LLMs:** code completion from the cloud, AI chats, cloud search.
- **Push documentation:** in-app "latest" docs, online tutorials, package readmes at HEAD.
- **Live analytics & dashboards:** crash telemetry viewers, cloud perf charts, remote logs.
- **Discovery feeds:** store/npm/pip search, template galleries, "examples from web".

Offline alternatives

- Local versioned backups (external SSD + scheduled robocopy/rsync).
- On-device models or no-AI; local snippets/library of patterns.
- Frozen PDF/HTML doc shelves with a local index page.
- Log files written to disk; simple local dashboards (static HTML).
- Curated example packs in 03_inputs\examples\.

Triage rule

- If removing feature X **does not change**: (a) what you can build, (b) what you can prove, or (c) your ability to restore—then X is a **comfort**. It belongs in the "reconnect later if desired" bucket.

ERL guidance

- ERL-0…3 must not rely on comforts at all.
- ERL-4…5 may reintroduce comforts **only** through export/import of artifacts, never as runtime prerequisites.

3.2 Essentials (actually required): activation, updates, revocation lists

Definition: Without these, your system may refuse to install, run, or trust core components—even offline. Essentials must be satisfied **without** a live network.

Typical essentials and offline strategies

- **OS activation / licensing:**
    - Prefer SKUs that support **offline or phone activation**.
    - Store proof of activation (confirmation IDs, screenshots, dates) in your binder.
    - Avoid software with "periodic online revalidation" requirements for the proof box.
- **Cumulative updates / service packs:**
    - Mirror **full offline installers** (.msu/.cab, offline ISO, Linux repo snapshots).
    - Apply during golden-image creation; keep the exact update set in 01_kits\updates\.
    - Record the update baseline in system_manifest.txt.
- **Code-signing roots and revocation lists (CRL/OCSP):**
    - Frozen trust: keep a **pinned, dated** root bundle used to verify the artifacts you mirrored.
    - Do **not** introduce new signed binaries after cut-off that you cannot validate offline.
    - Log the trust epoch: trust_epoch := yyyy-mm-dd for which your signatures and certs were verified.
- **Device drivers & firmware:**
    - Vendor-provided **standalone packages** with checksums; no web portals at install time.
    - Keep last-known-good versions; avoid forced auto-update mechanisms.
- **Toolchain completeness:**
    - Compilers, linkers, headers, libs, and all build tools must exist as **local artifacts**; builds must succeed with NICs physically absent.

    - Lockfiles must pin exact versions; no floating “latest”.

Red flags (convert to essentials or remove)

- “You must sign in to continue.” (identity gate for a local feature)
- “We’ll finish setup after you connect.” (OOBE dependency)
- “Cannot verify certificate validity” leading to block (runtime CRL/OCSP hard-fail)
- “Fetching dependencies…” without an offline override.

Acceptance tests

- **A1:** Cold build test with NICs disabled; passes 3x reproducibly.
- **A2:** Time-shift ±180 days; system still boots, runs toolchain, verifies existing signatures.
- **A3:** Bare-metal restore from your offline image; activation status preserved; drivers load; proofs run.

ERL guidance

- ERL-1: OOBE + activation satisfied offline.
- ERL-2: Updates, drivers, firmware in kit; time policy declared.
- ERL-3: Full toolchain and proofs succeed entirely from local mirrors.
- ERL-4: Bare-metal recovery image validated.
- ERL-5: Multi-year sustainment plan (keys, docs, installers) inventoried.

3.3 Ethics: vendor paternalism vs user sovereignty

Observation: Many “online by default” flows are rationalized as safety, simplicity, or “best experience.” Sometimes true; often paternalism. An exoplanet-ready posture makes the value judgment explicit.

Principles

- **Right to compute offline:** A legitimate, safety-preserving mode—especially for research, security, and pedagogy.
- **Informed consent:** Telemetry and cloud ties must be opt-in with clear local alternatives.

- **Composability over custody:** Provide documented, offline installers and manuals; let users compose their stack without surrendering identity or data.

Vendor minimums (ethical baseline)

1. **Local account path** that is first-class, documented, and stable across releases.
2. **Offline installers** for OS, SDKs, and drivers with published hashes.
3. **Static docs** (PDF/EPUB/CHM) matching the versions shipped.
4. **Deterministic builds**: no hidden online fetch in default project templates.
5. **No coercion UX**: don't hide the offline choice behind dark patterns.

User responsibilities (sovereignty with stewardship)

- Keep an **audit trail** (hashes, manifests, run logs).
- Practice **minimal trust**: verify signatures, freeze dependencies, and document the trust epoch.
- Prefer **reproducibility** over novelty: "known good" beats "latest."
- Respect **license terms**—choose tools that honor offline operation rather than bypassing restrictions.

A simple triage algorithm (run for every feature F)

1. Does F gate a core function (boot/install/build/prove/restore)?
    - If yes, classify F as **essential**; secure an **offline** path or replace the component.
2. If no: does F leak identity/telemetry or require continuous connectivity?
    - If yes, classify as **comfort** and disable by default; consider later via export/import only.
3. Else: **comfort**; document and move on.

Ethical outcome

- You end with a **deliberate perimeter**: essentials satisfied through verifiable offline means; comforts either excised or reintroduced **on your terms**. That is sovereignty—quiet, testable, and durable.

4) Designing an exoplanet-ready workstation

4.1 Hardware choices and BIOS/UEFI settings that favor autonomy

Pick hardware that doesn't *need* the internet to be "whole."

- **CPU/RAM/SSD:** Favor x86_64 with ample headroom (8C/16T, 32–64 GB, 1–2 TB SSD). Prefer SSDs with vendor-supplied **USB-bootable** firmware updaters and public checksums.
- **I/O:** Two **separate** USB-A sticks (ferry, quarantine). Optional third for recovery media. HDMI and a dumb keyboard—no driver drama.
- **Networking:** Built-in NIC/Wi-Fi is fine, but plan to **disable** them (firmware + OS).

UEFI/BIOS checklist (write down every change):

- **Disable NIC/Wi-Fi/Bluetooth.** (Air-gap starts in firmware.)
- **Secure Boot:**
    - If you'll use only stock-signed OS/toolchains, leave **Enabled**.
    - If you will boot custom WinPE/unsigned tools, set to **Enabled with custom keys** or **Temporarily Disabled** during maintenance. Log the state.
- **TPM:** Leave **Enabled** unless it blocks offline restore workflows. (BitLocker not recommended on an air-gap unless you have key management printed and tested.)
- **Virtualization:** Enable (SVM/VT-x) if you'll run local VMs for rebuild tests.
- **Boot order:** 1) USB, 2) Internal SSD, 3) Disabled others. Turn off network/PXE boot.
- **Power:** Disable "Fast Boot" (it can skip USB init). Keep deterministic fan curve to reduce thermal variance across runs.
- **Time:** Confirm RTC works; set a sane baseline date before first OS install.

ERL gate: You can reach installer + WinPE tools with all radios off and NICs absent from the boot list.

4.2 Filesystems, partitioning, and backup images that restore offline

Windows layout (clean, boring, reproducible):

- **GPT partitions** (sizes are suggestions):
    - ESP (FAT32) 512 MB → EFI boot
    - MSR 16 MB → Microsoft reserved
    - **OS (NTFS)** 200–400 GB → C:\ (OS + apps + toolchains)
    - **DATA (NTFS)** remaining → D:\ (proofbox\ tree, runs, docs)
    - Recovery 1–2 GB → WinRE tools
- **Why split?** Lets you reimage C: fast while preserving D:\proofbox\ manifests/artifacts.

USBs

- **Ferry:** exFAT (interop, large files), volume label FERRY_A.
- **Quarantine:** NTFS (for ACLs/ADS), label QUAR_Q.

Imaging (built-in, offline)

- **Capture a golden OS image (WinPE + DISM):**
    1. Boot WinPE from your Recovery USB.
    2. dism /capture-image /imagefile:D:\images\golden_os.wim /capturedir:C:\ /name:"golden_os" /compress:max /checkintegrity
    3. Hash it: Get-FileHash D:\images\golden_os.wim -Algorithm SHA256 >> D:\images\hashes.txt
- **Restore test (offline):**
    1. From WinPE: quick-format C:.
    2. dism /apply-image /imagefile:D:\images\golden_os.wim /index:1 /applydir:C:\
    3. bcdboot C:\Windows /s S:\EFI /f UEFI (S: = ESP in WinPE)
    4. Reboot; verify activation state, drivers, and that D:\proofbox\ is unchanged.
- **Data snapshots (robust, simple):**
    - robocopy D:\proofbox E:\snapshots\proofbox_YYYYMMDD /MIR /R:1 /W:1 /XJ /FFT /NP /LOG+:E:\snapshots\logs\proofbox_YYYYMMDD.log

- Hash index after snapshot:
  Get-ChildItem -Recurse D:\proofbox | Get-FileHash -Algorithm SHA256 | Out-File E:\snapshots\proofbox_YYYYMMDD.sha256

Linux note: Mirror scheme with EFI, ext4 for / and /data, plus offline partclone or fsarchiver images; store sha256sum catalogs.

ERL gate: Bare-metal restore of C: from WinPE + DISM succeeds with radios off; D:\proofbox\ verifies against last manifest.

4.3 Deterministic builds: pinning compilers, libs, and flags

Goal: Given the same sources + locks, produce bit-identical (or cryptographically stable) outputs across runs.

Pin the environment

- **Toolchain versions live in locks/** (text files you can read/print):
  - python.lock (exact version + embedded wheels set)
  - msvc.lock or llvm.lock (compiler build number)
  - rust.lock (rustup toolchain channel date, cargo vendor snapshot)
  - dotnet.lock (dotnet --info text dumped and signed)
- **PATH order:** Put your pinned tool dirs **first**; disable auto-shims.
- **Locale/encoding:** chcp 65001 or set UTF-8 system-wide. Prefer ASCII source where possible.
- **Timestamps & file paths:**
  - Use SOURCE_DATE_EPOCH where supported (C/C++/Rust/Go/Python packaging).
  - Strip absolute build paths (-ffile-prefix-map=Z:=. in clang/gcc; MSVC /FC-equivalent is to avoid embedding full paths).
  - Set a stable temp dir (e.g., C:\proofbox\tmp) and clean it each run.

Python (example)

- **Freeze wheels on the online box:**
  pip download -r locks\requirements.txt -d 02_toolchain\wheels
  Record: python --version, pip --version, pip list --format=freeze → locks\python.lock

- **Install offline:**
  pip install --no-index --find-links 02_toolchain\wheels -r locks\requirements.txt
- **Determinism knobs:**
  - set PYTHONHASHSEED=12345
  - Prefer wheels over sdists (compiles can vary). If compiling extensions, pin MSVC/LLVM and use fixed flags.

C/C++

- Choose **one** compiler family and pin it (MSVC XX.Y.Z or LLVM N).
- Prefer **/MT** (static CRT) on Windows for hermetic binaries (know the trade-offs).
- Disable non-deterministic features:
  - Remove build timestamps from banners (/Brepro in MSVC, -Wl,--build-id=sha1 style on ELF with reproducible flags if cross-building).
  - Avoid __DATE__, __TIME__. If needed, define them to constants via /D.

Rust

- cargo vendor → include vendor/ in repo.
- .cargo/config.toml:
- [source.crates-io]
- replace-with = "vendored-sources"
- [source.vendored-sources]
- directory = "vendor"
- [build]
- rustflags = ["-C", "debuginfo=0", "-C", "link-arg=/Brepro"]
- Build with --locked --offline.

Build script skeleton (Windows, PowerShell)

$env:PYTHONHASHSEED = "12345"

$env:SOURCE_DATE_EPOCH = "1732406400" # 2024-11-24 00:00:00 Z (example)

```
# Clean temp
Remove-Item -Recurse -Force C:\proofbox\tmp\* -ErrorAction SilentlyContinue | Out-Null
New-Item -ItemType Directory C:\proofbox\tmp | Out-Null
$env:TEMP = "C:\proofbox\tmp"
$env:TMP  = "C:\proofbox\tmp"

# Record tool versions
& python --version        | Tee-Object 04_runs\cur\logs\tool_versions.txt -Append
& pip --version           | Tee-Object 04_runs\cur\logs\tool_versions.txt -Append
& cl 2>&1                 | Tee-Object 04_runs\cur\logs\tool_versions.txt -Append
& rustc --version         | Tee-Object 04_runs\cur\logs\tool_versions.txt -Append

# Build
& python build.py         | Tee-Object 04_runs\cur\logs\build.txt
Get-ChildItem 04_runs\cur\outputs -Recurse |
  Get-FileHash -Algorithm SHA256 |
  Out-File 04_runs\cur\manifest.sha256
```

Verification loop

- Build the same target **3x** after reboot; compare hashes. Investigate any drift (timestamps, path salts, non-pinned libs).

ERL gate: Two independent cold builds (same locks) produce identical manifests (hash/size/order match).

4.4 Local time discipline: manual NTP stand-in, monotonic time notes

Why it matters: Certificates, signatures, logs, and some compilers bake in time. With no NTP, *you* are the time authority.

Define a time policy (print it and follow it)

- **Reference:** A single "time master" (e.g., rugged wristwatch or a second offline PC you set carefully).
- **Set cadence:** Set system time **once per workday/session**, not continuously.
- **Record:** Each session start → write time_log.txt entry:
  2025-09-24T19:58 local | set by: Doug | ref: Seiko | delta: +00:02:15

Windows controls

- Disable scheduled sync:
- Stop-Service W32Time
- Set-Service W32Time -StartupType Disabled
- reg add "HKLM\SYSTEM\CurrentControlSet\Services\W32Time\TimeProviders\NtpClient" /v Enabled /t REG_DWORD /d 0 /f
- Set time by hand (example, 24 Sep 2025 20:00 CDT):
  Set-Date -Date "09/24/2025 20:00:00"
- Stamp your run with both **local** and a **declared epoch** for reproducibility:
  - Local wall time in logs.
  - SOURCE_DATE_EPOCH fixed for build determinism (tied to your trust epoch).
    Note: Do **not** backdate after signing artifacts—monotonicity > prettiness.

Monotonic clock for measurements

- Use APIs that read a monotonic counter (Python time.perf_counter(); PowerShell [System.Diagnostics.Stopwatch]::GetTimestamp()), not wall clock, for benchmarking.
- Log both:
  t0_local, t0_perf, t1_perf, elapsed = t1_perf - t0_perf.

Time drift sanity checks

- **Drift ledger:** once a week, compare system clock vs reference; append delta.
- **Time-jump test:** add +180 days, run a small build/proof; everything should still verify (because you verified signatures at intake and pinned trust epoch). Revert time and log the test.

ERL gate: You can (a) set/record time manually, (b) build/prove with or without forward time shifts, and (c) your artifacts' verification does **not** depend on external time services.

Section 4 "ready" checklist (quick run)

- BIOS radios off; Secure Boot state documented.
- GPT partitions created; C: image captured; D:\proofbox\ populated.
- Recovery USB boots; DISM restore works from image; hashes match.
- Locks exist for each toolchain; env vars set; path order pinned.
- Three builds in a row → identical hashes.
- Time policy printed; W32Time disabled; run logs show local+monotonic stamps.

When you're ready, I can expand Section 5 (USB shuttle SOP) with exact command blocks for hashing, detached signing, quarantine→stage→release moves, and ingress/egress manifests.

5) The USB shuttle protocol (safe sneaker-net)

5.1 Roles: ferry, quarantine, staging, release

Drives (name them once, stick to it)

- **FERRY_A** (exFAT): goes **between** online and offline. Treated as "dirty." Prefer a stick with a **read-only hardware switch**; keep it **RO when entering** the offline room.
- **QUAR_Q** (NTFS): lives **only** on the offline box. First landing zone. Never execute from here.
- **OFFBOX_R** (NTFS or exFAT): optional "return" stick used only to export signed artifacts back out.

Folders on the offline box

D:\proofbox\

90_quarantine\ # from FERRY_A (NEVER EXECUTE HERE)

91_stage\ # verified inputs awaiting release

01_kits\ # installers, drivers (read-only)

02_toolchain\ # compilers, wheels, vendor trees

03_inputs\ # problem sets, examples

04_runs\<run_id>\ # outputs + logs + manifests

keys\ # public keys only; private key stays offbox

logs\ # ingress.log, egress.log, custody.log

bin\ # sha256sum.exe, minisign.exe, helper scripts

Human roles

- **Ferry operator (online):** Collect → hash → sign → copy to FERRY_A (RO off when writing, RO on to transport).
- **Quarantine officer (offline):** Copy from FERRY_A → 90_quarantine → verify hashes & signatures → promote to 91_stage or delete.
- **Release manager (offline):** Move from 91_stage → {01_kits, 02_toolchain, 03_inputs} + log the release.
- **Archivist (either side):** Periodically snapshot logs and manifests; print critical ones.

Golden rules

- Never execute or open files in **90_quarantine**. Promotion happens **only** after verify.
- Never plug **FERRY_A** into any third machine.
- Never bring private signing keys into the offline room.

5.2 Hash and signature routine (sha256 + detached sig)

Key model (simple and strong)

- Use **minisign** (Ed25519) or **GPG**; below uses minisign for brevity.
- **Private key**: generated and stored **offline on a separate, never-shuttled USB** (LOCKED in a safe). You sign **on the online box** by temporarily unlocking that USB, or you keep a **signing laptop** that never touches the internet (preferred).
- **Public key**: copy once to D:\proofbox\keys\minisign.pub (text file).

Online box: prepare an "intake bundle"

1. Put files to ship in a clean folder, e.g., C:\intake\2025-09-24_pack\.
2. Create a manifest and hashes:
3. cd C:\intake\2025-09-24_pack
4. dir /s /b > MANIFEST_FILELIST.txt
5. powershell -NoP "Get-ChildItem -Recurse | Get-FileHash -Algorithm SHA256 |
6. ForEach-Object { '{0} {1}' -f $_.Hash, $_.Path } |
7. Set-Content MANIFEST_SHA256.txt"
8. Sign both the **entire folder** (as a .zip) **and** the **hash list**:
9. powershell -NoP "Compress-Archive -Path * -DestinationPath ..\2025-09-24_pack.zip -Force"
10. minisign -S -s X:\private\minisign.key -m ..\2025-09-24_pack.zip
11. minisign -S -s X:\private\minisign.key -m MANIFEST_SHA256.txt

(Replace X: with the removable that holds your private key.)

12. Copy to **FERRY_A** (write-enable switch ON while copying):
13. robocopy C:\intake\2025-09-24_pack F:\2025-09-24_pack /E /R:1 /W:1
14. copy C:\intake\2025-09-24_pack.zip F:\
15. copy C:\intake\2025-09-24_pack.zip.minisig F:\

Flip **FERRY_A** to **read-only** before leaving the online room.

Offline box: verify and promote

1. Copy from **FERRY_A** to **90_quarantine**:
2. robocopy F:\2025-09-24_pack D:\proofbox\90_quarantine\2025-09-24_pack /E /R:1 /W:1
3. copy F:\2025-09-24_pack.zip D:\proofbox\90_quarantine\
4. copy F:\2025-09-24_pack.zip.minisig D:\proofbox\90_quarantine\
5. Verify signatures using the **public key**:
6. cd D:\proofbox\90_quarantine
7. minisign -Vm 2025-09-24_pack.zip -P "D:\proofbox\keys\minisign.pub"
8. minisign -Vm .\2025-09-24_pack\MANIFEST_SHA256.txt -P "D:\proofbox\keys\minisign.pub"

Expect “Signature and comment signature verified.”

9. Verify hashes (bit-for-bit):
10. cd D:\proofbox\90_quarantine\2025-09-24_pack
11. powershell -NoP "$m = Get-Content MANIFEST_SHA256.txt |
12. ForEach-Object {
13. $hash,$path = $_.Split(' ',2)
14. $got = (Get-FileHash $path -Algorithm SHA256).Hash
15. if ($got -ne $hash) { Write-Host 'MISMATCH: ' $path; exit 2 }
16. }; Write-Host 'ALL OK'"
17. **Promote** (no edits, preserve timestamps):
    - If installers/tooling → D:\proofbox\02_toolchain_inbox\2025-09-24_pack\
    - If drivers/firmware → D:\proofbox\01_kits\drivers\2025-09-24_pack\
    - If inputs/problems → D:\proofbox\03_inputs\2025-09-24_pack\
18. robocopy D:\proofbox\90_quarantine\2025-09-24_pack D:\proofbox\91_stage\2025-09-24_pack /E /COPY:DAT /DCOPY:T
19. **Release** (record in ingress log first, see 5.4), then:

20. robocopy D:\proofbox\91_stage\2025-09-24_pack D:\proofbox\02_toolchain\wheels /E /MOV

Tip: Keep minisign.exe and sha256sum.exe in D:\proofbox\bin\ and always call them via absolute path in scripts.

5.3 Cold-scanning and content policies (no macros, no autorun)

Cold-scan concept: With no fresh AV signatures offline, you enforce **format hygiene** and **execution discipline** instead.

Execution discipline

- **Disable AutoRun/AutoPlay**:
- reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer /v NoDriveTypeAutoRun /t REG_DWORD /d 255 /f
- reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer /v NoDriveTypeAutoRun /t REG_DWORD /d 255 /f
- Never run binaries **from** 90_quarantine or 91_stage. Only from curated locations (01_kits, 02_toolchain).
- Prefer **static installers** over "web installers."

Document & code hygiene

- **Office docs:** Only accept **.pdf** (prefer PDF/A). Reject .docm/.xlsm and any embedded macro content. If you must accept .docx, convert to .pdf on the **online** side before signing.
- **Archives:** Accept **.zip** only; reject self-extracting .exe archives. Disallow nested archives >1 level deep. No files with double extensions (.pdf.exe).
- **Scripts:** Accept **.ps1, .py, .sh** only into 02_toolchain after review; require cleartext shebang/headers; forbid obfuscation/minified one-liners.
- **Binaries:** Only accept binaries built by **your** toolchain or signed by **you**. Third-party EXEs are limited to vendors you already whitelisted in 01_kits, each with vendor checksum.

Static scanning (lightweight, offline)

- Run **sigcheck** (Sysinternals) offline to list signatures, timestamps, entropy hints:
- sigcheck64.exe -q -m -h D:\proofbox\90_quarantine\2025-09-24_pack > ..\..\logs\sigcheck_2025-09-24.txt
- Simple “bad patterns” grep (PowerShell) for scripts:
- Get-ChildItem -Recurse *.ps1,*.vbs,*.js | ForEach-Object {
- $t = Get-Content $_.FullName -Raw
- if ($t -match "FromBase64String|IEX|New-Object Net\.WebClient|DownloadString") {
- Write-Host "FLAG: " $_.FullName
- }
- }
- For archives, list contents and check paths for traversal:
- powershell -NoP "Add-Type -A System.IO.Compression.FileSystem;
- $z=[IO.Compression.ZipFile]::OpenRead('pack.zip');
- $z.Entries | % { $_.FullName } | Set-Content pack_contents.txt"
- # Review pack_contents.txt for ..\ or absolute paths

Policy: convert on the online side

- Convert **docs to PDF/A**.
- Convert **images to PNG/JPG** (no embedded scripts/metadata).
- Vendor **source tarballs**; avoid running vendor build scripts that fetch the internet.

5.4 Chain-of-custody log you can actually keep

Three logs (plain CSV)

- D:\proofbox\logs\custody.log — movements of media (who, what, when).
- D:\proofbox\logs\ingress.log — every intake bundle verified and released.
- D:\proofbox\logs\egress.log — every artifact leaving the box.

Formats (CSV, no commas in paths)

- **custody.log**
- ts_local,actor,media,action,location,notes
- 2025-09-24T19:55,Doug,FERRY_A,RO->RW,OnlineDesk,"prep copy"
- 2025-09-24T20:05,Doug,FERRY_A,RW->RO,OnlineDesk,"sealed for transport"
- 2025-09-24T20:20,Doug,FERRY_A,insert,OfflineLab,"verify pack 2025-09-24"
- 2025-09-24T20:45,Doug,FERRY_A,remove,OfflineLab,""
- **ingress.log**
- ts_local,pack_name,src_media,bytes,files,sha256_zip,signer_pubkey_short,result,released_to
- 2025-09-24T20:30,2025-09-24_pack,FERRY_A,18344219,142,ABCD...9912,minisign:Zy7N..kmP,OK,02_toolchain\wheels
- **egress.log**
- ts_local,run_id,artifact,bytes,sha256,sig,media,notes
- 2025-09-24T22:10,2025-09-24_2200,outputs\proofset_P1.zip,4420192,9F1A...77C1,OK,OFFBOX_R,"deliver for publish"

One-button helpers (drop these scripts in D:\proofbox\bin\)

- **Compute and sign a run manifest (offline)**
- # file: make_run_manifest.ps1
- param(
- [string]$RunDir = "D:\proofbox\04_runs\cur",
- [string]$PubKey = "D:\proofbox\keys\minisign.pub",
- [string]$PrivKeySig = "" # empty offline; you will sign on the online box
-)
- $out = Join-Path $RunDir "manifest.sha256"

- Get-ChildItem -Recurse (Join-Path $RunDir "outputs") |
- Get-FileHash -Algorithm SHA256 |
- ForEach-Object { "{0} {1}" -f $_.Hash, $_.Path } |
- Set-Content $out
- if ($PrivKeySig -ne "") {
- & D:\proofbox\bin\minisign.exe -S -s $PrivKeySig -m $out
- }
- Write-Host "Wrote $out"
- **Verify an intake pack and append to ingress.log**
- # file: verify_and_ingress.ps1
- param(
- [string]$PackDir = "D:\proofbox\90_quarantine\2025-09-24_pack",
- [string]$ZipFile = "D:\proofbox\90_quarantine\2025-09-24_pack.zip",
- [string]$PubKey = "D:\proofbox\keys\minisign.pub",
- [string]$ReleaseTo = "D:\proofbox\02_toolchain\wheels"
-)
- $ts = Get-Date -Format "yyyy-MM-ddTHH:mm"
- $bytes = (Get-Item $ZipFile).Length
- $files = (Get-ChildItem -Recurse $PackDir | Measure-Object).Count
- $sha = (Get-FileHash $ZipFile -Algorithm SHA256).Hash
- & D:\proofbox\bin\minisign.exe -Vm $ZipFile -P $PubKey | Out-Null
- & D:\proofbox\bin\minisign.exe -Vm (Join-Path $PackDir "MANIFEST_SHA256.txt") -P $PubKey | Out-Null
- robocopy $PackDir $ReleaseTo /E /COPY:DAT /DCOPY:T | Out-Null
- "$ts,$((Split-Path $PackDir -Leaf)),FERRY_A,$bytes,$files,$sha,minisign,OK,$ReleaseTo" |
- Add-Content D:\proofbox\logs\ingress.log

- Write-Host "Ingested and released to $ReleaseTo"

Periodic hardening

- **Print** the first and last page of each log weekly; sign the printouts by hand.
- **Snapshot** D:\proofbox\logs\ to a read-only archive on **OFFBOX_R** with hashes.
- **Rotate** FERRY_A annually; physically destroy old sticks.

Acceptance test for Section 5 (15 minutes)

1. Prepare a tiny intake pack online (one .txt and one .pdf). Hash + sign.
2. Shuttle with **FERRY_A (RO on)**; copy to **90_quarantine**.
3. Verify sigs + hashes; promote to **91_stage**; release to **03_inputs**.
4. Confirm entries in **custody.log** and **ingress.log**.
5. Run a toy job; create **manifest.sha256**; export to **OFFBOX_R**; log in **egress.log**.

Pass if no execution occurs in quarantine, all verifications succeed, and your logs read like a flight recorder—plain, boring, trustworthy.

6) Packaging an offline universe

6.1 Mirroring core docs: language specs, manpages, PDFs

Goal: A self-contained bookshelf at D:\proofbox\00_docs\ that never needs the web.

Layout

```
00_docs\
  index.html          # your offline start page
  specs\              # language and tool specs (PDF/EPUB/HTML)
  manuals\            # OS, shell, compilers, linkers
  apidocs\            # generated local API docs (Sphinx, cargo doc, Doxygen)
  playbooks\          # your SOPs: OOBE, imaging, shuttle, builds
  cheatsheets\        # quick refs (one pagers, ASCII-safe)
  manifest\           # docs_manifest.txt + sha256 list + signatures
```

Process

1. **Collect stable copies** on the online box (prefer PDF/EPUB). Avoid "latest"; pick versions that match your toolchain.
2. **Normalize filenames** (no spaces, add version tags): python_3.12.5_ref.pdf, msvc_19.40_flags.pdf.
3. **Hash and sign** the lot; ship via shuttle; land under 00_docs\.
4. **Generate local API docs** for your own code and pinned libs (examples):
    - Python: pydoc -w yourpkg or Sphinx make html → copy to apidocs\yourpkg\.
    - Rust: cargo doc --offline --no-deps → target\doc\ → apidocs\rust\.
    - C/C++: Doxygen to HTML → apidocs\cxx\.

Make the shelf usable

- Build a simple index.html with sections and links to local files (file:///D:/proofbox/00_docs/...).
- Add a **"Where to look first"** page (2 screens max) that routes common questions to the right PDF/HTML.

Acceptance

- Pull the network, reboot, open index.html, and navigate to: OS manual → Python ref → your SOP PDF → Rust docs → Doxygen. All must open locally.

6.2 Mirroring package repos and wheels; freezing lockfiles

Goal: Install and build everything from your own mirror directories, not the internet.

Layout

```
02_toolchain\
  wheels\              # Python wheels tarpit
  rust\vendor\         # cargo vendor snapshot
  node\registry\       # npm/pnpm tarballs or pnpm store
  dotnet\layout\       # VS/.NET offline layout
```

cxx\libs\ # zipped source or prebuilt static libs

locks\ # exact versions (text lockfiles)

manifests\ # sha256 of everything here + signatures

Python (example)

- Online (collect):
- cd C:\kit\python
- copy locks\requirements.txt .
- python -m pip download -r requirements.txt -d wheels\
- python -V > locks\python.lock
- pip --version >> locks\python.lock
- pip freeze -r requirements.txt >> locks\python.lock
- Offline (install):
- setx PIP_NO_INDEX 1
- setx PIP_FIND_LINKS "file:///D:/proofbox/02_toolchain/wheels"
- pip install --no-index --find-links D:\proofbox\02_toolchain\wheels -r D:\proofbox\02_toolchain\locks\requirements.txt

Rust

- Online:
- rustup toolchain install 1.80.1
- rustup default 1.80.1
- cargo vendor --versioned-dirs --locked --respect-source-config --sync ./Cargo.lock ./vendor
- # Save toolchain info
- rustc --version > locks\rust.lock
- cargo --version >> locks\rust.lock
- Offline (.cargo\config.toml in project or global):
- [source.crates-io]

- replace-with = "vendored-sources"
- [source.vendored-sources]
- directory = "D:/proofbox/02_toolchain/rust/vendor"

Build with: cargo build --locked --offline

Node

- Prefer **pnpm** with an offline store:
    - Online: pnpm store path → copy store to 02_toolchain\node\store\
    - Save pnpm-lock.yaml to locks\.
    - Offline .npmrc/.pnpmfile.cjs points cache/store to file:///D:/proofbox/02_toolchain/node/...
- Or archive exact tarballs:
- npm pack <pkg>@<ver> --pack-destination .\registry\
- # repeat for all pkgs in package-lock.json

.NET / Visual C++

- Use **Visual Studio / .NET "layout"** features on the online box to create an offline cache:
    - VS Build Tools: vs_buildtools.exe --layout C:\kit\dotnet\layout --lang en-US
    - .NET SDK: official offline installers copied to dotnet\layout\
- Save dotnet --info to locks\dotnet.lock.

C/C++ libs

- Vendor **source tarballs** or **prebuilt static libs** with matching headers. Avoid build scripts that fetch during configure.
- Write a tiny cxx.lock listing compiler version, CRT mode (/MT or /MD), and each lib version/hash.

Freeze

- For each ecosystem, the "lock" file is king. Disallow floating versions (*, ^, ~, latest).

Acceptance

- Physically remove NICs (or disable in BIOS) and perform a **clean rebuild** of your smallest target per ecosystem using only 02_toolchain\ + locks\. Zero network touches.

6.3 Building local indices (search without the web)

Goal: Type a term and find it in your shelf and code docs—all offline.

Options (pick one simple path and stick with it)

- **Static HTML index with Lunr-style search (offline JS)**
    - Save lunr.min.js (or tiny search lib) to 00_docs\search\.
    - Generate a small JSON index (docs_index.json) listing title, path, and a short excerpt per doc.
    - Hook a search box in index.html that filters and links to file:// targets.
- **Minimal "grep portal" (fast, no JS)**
    - Build a **content map**: docs_map.txt where each line = title|path|keywords.
    - Add a PowerShell helper:
    - # search_docs.ps1
    - param([string]$q)
    - Get-Content D:\proofbox\00_docs\manifest\docs_map.txt |
    - Where-Object { $_ -match $q } |
    - ForEach-Object { $_ }
    - Bind it to a desktop shortcut; results show local paths to open.

Build the index (PowerShell snippet)

```
# index_docs.ps1

$root = "D:\proofbox\00_docs"

$docs = Get-ChildItem $root -Recurse -Include *.pdf,*.html,*.epub

$lines = @()

foreach ($d in $docs) {
```

```
  $rel = $d.FullName.Replace($root + "\","")
  $title = [IO.Path]::GetFileNameWithoutExtension($d.Name)
  $kw = ($rel -replace "\\"," ").ToLower()
  $lines += "$title|$rel|$kw"
}
$manifest = Join-Path $root "manifest\docs_map.txt"
$lines | Set-Content $manifest
```

Index integrity

- Hash the index and docs directory:
- Get-ChildItem -Recurse D:\proofbox\00_docs | Get-FileHash -Algorithm SHA256 > D:\proofbox\00_docs\manifest\docs_sha256.txt
- Sign docs_sha256.txt on the online signer; store the .minisig next to it.

Acceptance

- With no network, search for a term that exists in a PDF title and an API HTML page; both open locally from the start page or via the helper.

6.4 Teaching the toolchain to prefer “file://” over “https://”

Goal: All package/tools look **first** to your local mirrors—and never try the network.

Python

- Environment:
- setx PIP_NO_INDEX 1
- setx PIP_FIND_LINKS "file:///D:/proofbox/02_toolchain/wheels"
- pip.ini (optional, %APPDATA%\pip\pip.ini):
- [global]
- no-index = yes
- find-links = file:///D:/proofbox/02_toolchain/wheels

Rust / cargo

- .cargo\config.toml:
- [net]
- offline = true
- [source.crates-io]
- replace-with = "vendored-sources"
- [source.vendored-sources]
- directory = "D:/proofbox/02_toolchain/rust/vendor"

Node

- .npmrc or .pnpmrc (example pnpm):
- offline = true
- store-dir = D:\proofbox\02_toolchain\node\store
- prefer-offline = true
- For npm tarballs mirror:
- registry = "file:///D:/proofbox/02_toolchain/node/registry"
- audit = false
- fund = false

.NET / VS build tools

- Use the **layout** cache path consistently; install SDK/workloads from your layout directory.
- NuGet NuGet.Config:
- <configuration>
- <packageSources>
- <add key="local" value="D:\proofbox\02_toolchain\nuget\" />
- <clear /> <!-- ensure only local -->
- </packageSources>

- <config>
- <add key="globalPackagesFolder" value="D:\proofbox\02_toolchain\nuget\packages" />
- </config>
- </configuration>

C/C++ and build systems

- **CMake**: set CMAKE_FIND_ROOT_PATH and vendor dependencies; disable FetchContent that hits the net.
- **MSBuild**: avoid tasks that download during build; vendor toolchains.

System-level guardrails

- Disable NICs in BIOS and Device Manager.
- Windows Firewall outbound default = **Block** (belt-and-suspenders).
- ETW or firewall logs left on; should remain quiet during builds.

Quick verifier

- Search for http/https in your project and tool configs:
- Get-ChildItem D:\proofbox -Recurse -Include *.toml,*.json,*.ini,*.config,*.ps1,*.bat,*.cmd |
- Select-String -Pattern "https?://" | Group-Object Path | Select-Object Name

Review and remove/replace any stray online endpoints.

Acceptance

- Cold boot with radios off → run pip install, cargo build --offline, pnpm install, dotnet restore (against your mirror). All succeed. No outbound attempts logged.

Section 6 “done” checklist

- 00_docs\index.html works offline; docs manifest + hashes signed.
- 02_toolchain\ contains mirrors for Python/Rust/Node/.NET/C++ with **locks**.
- A local search path exists (HTML search or grep portal).

- All package managers prefer file:// mirrors and succeed with NICs absent.
- A clean rebuild of a sample target in each stack passes with zero network use.

7) Operating without telemetry

7.1 What you lose: autocomplete from the hive mind

Cloud goodies you're skipping

- Cloud IDE copilots and large template galleries.
- Web search baked into editors ("quick fix from web", issue lookup).
- Crash/usage analytics dashboards.
- Online package discovery ("similar packages", "trending").
- Live vulnerability feeds and auto-patch prompts.

Offline substitutes (practical)

- **Local snippets + patterns:** keep D:\proofbox\03_inputs\snippets\ with vetted examples.
- **ctags/gtags & ripgrep:** jump-to-symbol and code search with zero network.
- **Local docs portal:** your 00_docs\index.html + PDF shelf + search_docs.ps1.
- **On-device linting/formatting:** pinned linters, no server calls.
- **Optional local models:** only if you want them—small on-device completions (no outbound). Treat as a *comfort*, not essential.

Net: you lose hive intelligence, you gain a quieter, more knowable loop.

7.2 What you gain: reproducibility, performance determinism, privacy

Reproducibility

- No surprise updates mid-run; lockfiles + mirrored artifacts mean same inputs each time.
- Stable toolchain versions; identical hashes across rebuilds become normal.

Performance determinism

- No background telemetry or updaters stealing cycles.

- Less jitter: identical power plan, identical flags, identical environment → tight runtime variance.

Privacy / attack surface

- No outbound payloads (code, titles, file paths) leaking.
- Drastically reduced ingress vectors: everything arrives via your signed shuttle.

Operational clarity

- Fewer moving parts → simpler failure modes → faster RCA (root cause analysis).
- Logs read like a flight recorder; comparisons across weeks are apples-to-apples.

7.3 Benchmarks: compile time, proof time, variance across runs

Metrics (define once, reuse)

- compile_time_ms: wall time to build a fixed target from a clean state.
- proof_time_ms: wall time for a fixed proof suite P with seed s.
- cpu_util_avg, cpu_util_peak: percent during run.
- mem_peak_mb: peak working set.
- io_read_mb, io_write_mb: logical I/O bytes.
- cv_pct: coefficient of variation across N runs = 100 * stddev / mean.

Pre-run controls (reduce noise)

- Power plan: **Ultimate Performance** (or High performance).
  powercfg -setactive SCHEME_MIN (check GUIDs; set once)
- Thermals: run with case fans steady; avoid big ambient swings.
- Background tasks: defender scans off for your working dirs; scheduled tasks paused.
- Clean caches consistently (e.g., delete build/, restart between runs if needed).
- Seeds fixed: PYTHONHASHSEED=12345, deterministic flags set.

Bench harness (PowerShell; drop in D:\proofbox\bin\bench.ps1)

```
param(
  [string]$Name = "compile_sample",
```

```
  [int]$N = 5,
  [scriptblock]$Prep = { & git clean -fdx >$null 2>&1 },
  [scriptblock]$Cmd  = { & python build.py },
  [string]$OutDir = "D:\proofbox\04_runs\bench"
)

# Helpers
function now_ms { [int64]([DateTimeOffset]::Now.ToUnixTimeMilliseconds()) }
function cpu_pct {
  (Get-Counter '\Processor(_Total)\% Processor Time' -SampleInterval 1 -MaxSamples 1
  ).CounterSamples[0].CookedValue
}
function mem_peak_mb($p){ [int]($p.PeakWorkingSet64/1MB) }

# Ensure dirs
$run_id = (Get-Date -Format 'yyyyMMdd_HHmmss')
$dir = Join-Path $OutDir "$Name`_$run_id"
New-Item -ItemType Directory $dir | Out-Null
$csv = Join-Path $dir "results.csv"
"idx,start_ms,stop_ms,elapsed_ms,cpu_start_pct,cpu_stop_pct,mem_peak_mb,io_read_mb,io
_write_mb,rc" | Out-File $csv

for($i=1; $i -le $N; $i++){
  & $Prep
  Start-Sleep -Seconds 2
  $p = Start-Process -FilePath "powershell.exe" -ArgumentList "-NoP","-Command",$Cmd -
PassThru -WindowStyle Hidden
```

```
  $start = now_ms
  $cpu0 = [Math]::Round((cpu_pct),2)
  $io_start = $p | Get-Process | Select-Object -ExpandProperty IOReadBytes,IOWriteBytes
  $sw = [System.Diagnostics.Stopwatch]::StartNew()
  $p.WaitForExit()
  $sw.Stop()
  $stop = now_ms
  $cpu1 = [Math]::Round((cpu_pct),2)
  $proc = Get-Process -Id $p.Id -ErrorAction SilentlyContinue
  $io_end = $proc | Select-Object -ExpandProperty IOReadBytes,IOWriteBytes
  $io_read_mb  = [Math]::Round((($io_end.IOReadBytes  - $io_start.IOReadBytes)/1MB),2)
  $io_write_mb = [Math]::Round((($io_end.IOWriteBytes - $io_start.IOWriteBytes)/1MB),2)
  $mem = if($proc){ mem_peak_mb $proc } else { 0 }

"$i,$start,$stop,$($sw.ElapsedMilliseconds),$cpu0,$cpu1,$mem,$io_read_mb,$io_write_mb,$(
$p.ExitCode)" | Add-Content $csv
  Start-Sleep -Seconds 3
}

# Summaries
$rows = Import-Csv $csv
$times = $rows | ForEach-Object { [double]$_.elapsed_ms }
$avg = [Math]::Round(($times | Measure-Object -Average).Average,2)
$sd  = [Math]::Round([Math]::Sqrt((($times | ForEach-Object { ($_-$avg)*($_-$avg) }) |
Measure-Object -Sum).Sum / ($times.Count)),2)
$cv  = if($avg -ne 0){ [Math]::Round(100.0*$sd/$avg,2) } else { 0 }
```

"mean_ms=$avg" | Out-File (Join-Path $dir "summary.txt")

"stddev_ms=$sd" | Add-Content (Join-Path $dir "summary.txt")

"cv_pct=$cv" | Add-Content (Join-Path $dir "summary.txt")

"n=$($times.Count)" | Add-Content (Join-Path $dir "summary.txt")

Write-Host "Wrote $dir"

Use it for compile and proof

- **Compile benchmark:**
- D:\proofbox\bin\bench.ps1 -Name "compile_core" -N 5 `
- -Prep { & cmd /c "rmdir /s /q build 2>nul & mkdir build & cd build" } `
- -Cmd { & cmake ..; & cmake --build . --config Release }
- **Proof benchmark (seeded):**
- $env:PYTHONHASHSEED="12345"
- D:\proofbox\bin\bench.ps1 -Name "proof_P0" -N 5 `
- -Prep { & powershell -NoP "Remove-Item -Recurse -Force .\out 2>$null; New-Item -ItemType Directory out | Out-Null" } `
- -Cmd { & python run_proofs.py --suite P0 --seed 12345 --out .\out }

Targets and thresholds (tune to your box)

- **Determinism:** identical output hashes across runs; mismatch = fail.
- **Variance:** cv_pct <= 5 for compile/proof times on a warmed system. Investigate if higher (thermal throttling, background tasks).
- **Regression watch:** keep a bench_history.csv (append from each summary.txt). Flag if mean_ms drifts > 10% without intentional changes.

Run log snippet (append to 04_runs\bench\bench_index.csv)

ts_local,name,mean_ms,cv_pct,n,seed,toolchain_hash

2025-09-24T21:40,compile_core,8421,2.7,5,-,msvc_19.40_py312.lock

2025-09-24T21:55,proof_P0,11234,3.2,5,12345,msvc_19.40_py312.lock

If jitter spikes

- Check CPU temp and clocks (throttling). Clean vents or reduce ambient.
- Confirm power plan. Disable any lingering updaters/telemetry.
- Verify disk health and free space.
- Re-seed and retry; re-image only if drift persists and root cause is unclear.

Acceptance for Section 7

- Both compile and proof benches run 5× with cv_pct <= 5 and identical artifact hashes.
- Bench summaries stored under 04_runs\bench\...; history shows stability over time.
- No outbound attempts recorded during any bench (firewall/ETW quiet).

That's a full telemetry-free operating profile: you know exactly what you lost, what you gained, and you can *measure* that the box is steady.

8) Exoplanet Readiness Levels (ERL)

ERL-0 — Boots offline

Goal: Machine reaches a usable local desktop/console with *zero* network.
Requirements

- All radios disabled in BIOS (LAN/Wi-Fi/BT) and in OS.
- Local admin account exists; no cloud sign-in anywhere in the path.
- Boot order set: USB -> SSD; PXE disabled.
  Evidence to keep
- oobe_transcript.txt (screens accepted/declined).
- Photo/screenshot of Device Manager showing NICs disabled.
- bios_settings.txt (manually transcribed or vendor export).
  15-minute acceptance

1. Physically unplug any Ethernet.
2. Power on, reach desktop.

3. Verify no “finish setup when online” banners; no outbound prompts.
   Failure modes

- OOBE forces sign-in; hidden local path.
- NIC re-enables via “smart” firmware defaults.
  Upgrade path to ERL-1
- Build proper install USB; capture a reference image that boots the same way every time.
- Gather offline cumulative updates and drivers.

ERL-1 — Installs + updates from local media

Goal: Fresh install + essential OS updates fully offline.
Requirements

- Offline install USB with your edition, local admin prepped (or captured image).
- Offline update set (e.g., .msu/.cab or distro ISO snapshot).
- Drivers for storage/GPU/chipset staged locally; no “search online”.
  Evidence to keep
- install_log.txt (time, media used, edition, key steps).
- post_install_baseline.txt (services, firewall, policies).
- drivers_inventory.csv (device PNP ID -> driver version).
  15-minute acceptance

1. Reimage C: from your USB.
2. Apply offline updates.
3. Reboot; check “unknown device” count = 0.
   Failure modes

- Web installers for drivers; Windows Update attempts.
- Update set incomplete (Servicing Stack mismatch).
  Upgrade path to ERL-2
- Build 00_docs/ shelf; mirror toolchains and lock versions; write a time policy.

ERL-2 — Full toolchain mirrors and docs local

Goal: You can read docs and install compilers/libs from file:// only.
Requirements

- 00_docs/ shelf populated; docs_manifest + sha256 present.
- 02_toolchain/ contains mirrors: Python wheels, cargo vendor, pnpm store or npm tarballs, .NET layout, C/C++ libs.
- Lockfiles exist and pin exact versions.
- Package managers configured: prefer local, no-index/offline flags set.
  Evidence to keep
- locks*.lock, manifests*.sha256, signatures.
- Screenshots or configs showing file:// mirror settings (pip.ini, .cargo/config, .npmrc, NuGet.Config).
  15-minute acceptance

1. With NICs disabled, pip install -r ... from wheels.
2. cargo build --locked --offline with vendored sources.
3. Open 3 docs (OS manual, language ref, your SOP) from index.html.
   Failure modes

- Hidden “fetch at build” steps.
- Floating versions in lockfiles; inconsistent mirrors.
  Upgrade path to ERL-3
- Add reproducible build flags; seed randomness; write a build harness and manifest signer.

ERL-3 — Deterministic builds + reproducible proofs

Goal: Same inputs -> bit-stable outputs and proof results, repeatedly.
Requirements

- Determinism controls: fixed seeds, fixed temp dir, pinned PATH order, stripped timestamps and absolute paths.
- Build/proof scripts emit manifest.sha256 and detached sigs.

- Bench harness logs mean/stddev and cv_pct; target cv_pct <= 5 for steady runs.
  Evidence to keep
- 04_runs\YYYYMMDD_HHMM\manifest.sha256 (3 runs, identical).
- summary.txt with mean_ms, stddev_ms, cv_pct; bench_history.csv.
- Tool versions dump (tool_versions.txt) per run.
  15-minute acceptance

1. Reboot, run the same build 3x.
2. Hash lists identical; bench cv_pct <= 5.
3. Proof suite P with fixed seed produces same artifact set/hash.
   Failure modes

- Compiler embeds build time or full paths.
- Background services create jitter.
  Upgrade path to ERL-4
- Capture a golden image; practice bare-metal restore; script hash verification of restored files.

ERL-4 — Recover from bare metal with no network

Goal: Full restore to a known-good state from your own media only.
Requirements

- WinPE (or equivalent) recovery USB; DISM (or partclone/fsarchiver) images for C:; scripted restore steps.
- D:\ (data) separated and snapshotted; post-restore validation script compares hashes to canonical manifest.
- Activation/licensing preserved or restorable offline.
  Evidence to keep
- Printed restore runbook; images\golden_os.wim sha256; proofbox_sha256.txt.
- recovery_test_log.txt with stopwatch start/end and result.
  15-minute acceptance

1. Wipe C:; apply golden_os.wim; bcdboot; reboot.

2. Open docs; run micro build; verify D:\proofbox\ hashes match manifest.
   Failure modes

- Bootloader not rebuilt; Secure Boot state mismatch.
- Drivers missing in image; activation lost.
  Upgrade path to ERL-5
- Build a sustainment kit: keys, firmware, drivers, installers, and a written time policy that lasts years.

ERL-5 — Multi-year sustainment (keys, drivers, firmware, time)

Goal: Operate for Y years offline (pick Y>=3) without hitting the net.
Requirements

- **Keys:** Offline signing key on a separate never-shuttled USB; public key in keys\. Printed fingerprint.
- **Firmware & drivers:** Last-known-good offline updaters with vendor hashes; printed table of device IDs -> versions.
- **Installers:** All OS updates, toolchains, SDKs mirrored with their own hashes.
- **Docs:** Frozen snapshots matching toolchain versions; index works offline.
- **Time:** Written time policy; w32time disabled; monotonic benchmarking; documented "trust epoch" for signatures.
- **Logs:** Custody/ingress/egress logs rotated and periodically printed.
- **Spare media:** Redundant recovery/ferry sticks; annual rotation plan.
  Evidence to keep
- inventory.txt listing versions, counts, media locations, expiry dates (if any).
- trust_epoch.txt declaring the cutoff date used to verify all shipped signatures.
- Printed: key fingerprint, restore runbook, hotline page with the 10 rules.
  15-minute acceptance

1. Simulate a "2 years later" time jump: set clock +730 days.
2. Build & run proofs; verify existing signatures and manifests still check out.

3. Apply one staged firmware update from your kit (if you carry any), then confirm the system still builds and proves.
   Failure modes

- Hidden time dependencies (license revalidation, CRL/OCSP hard-fail).
- Lost or unreadable recovery/signing media; undocumented key rotation.
  Sustainment cadence
- Quarterly: bare-metal restore drill + hash verify.
- Quarterly: bench run; append to bench_history.csv.
- Semiannual: print and sign the last pages of custody/ingress/egress logs.
- Annual: rotate ferry media; re-verify installer hashes; dust-test the recovery USB.

Quick ladder summary (what changes at each step)

- **ERL-0 -> 1:** From “this boots” to “this installs and patches offline.”
- **ERL-1 -> 2:** From “OS only” to “full local dev universe” (docs + mirrors + locks).
- **ERL-2 -> 3:** From “can build” to “builds and proofs are reproducible and measurable.”
- **ERL-3 -> 4:** From “steady state” to “can resurrect from nothing but our kit.”
- **ERL-4 -> 5:** From “survivable incident” to “multi-year autonomy with a paper trail.”

If you want, I can turn these into printable one-page checklists (one per level) and a single “ERL Scorecard” you can tape to the chassis and initial after each drill.

9) Case study: the Beelink proof device

9.1 OOBE to first proof: gotchas and workarounds (high-level)

Pre-flight (before power-on)

- Disconnect Ethernet, remove any Wi-Fi credentials; have only: keyboard, monitor, your **Install USB**, and your **Updates/Drivers USB**.
- In UEFI:
    - Disable LAN/Wi-Fi/BT, PXE/NIC boot.

- Boot order: 1) USB, 2) NVMe SSD, others disabled.
- Secure Boot: Enabled (stock keys) unless you need custom WinPE; note the state.
- Save and exit.

OOBE (Out-of-Box Experience) pitfalls

- Cloud sign-in loops: choose the local account path (Pro/Edu SKUs). If the screen buries it, select "Domain join instead" / "Offline account" variant. If that UI disappears on a future build, abort and deploy your **captured image** instead of wrestling the wizard.
- Forced network screen: keep radios off so it cannot "continue after connecting." If presented, pick "I don't have Internet" → "Continue with limited setup".

Post-install harden (10 minutes)

- Create your local admin cluster (or your standard name), set a long passphrase.
- Turn off Windows Time sync (services + policy) and set local time manually once (per your time policy).
- Rename the box (e.g., S5f), reboot.
- Disable Device Installation from Windows Update; force "Never" ask to search online for drivers.
- In Windows Firewall, set Outbound default = Block (you're already air-gapped—this is belt-and-suspenders).

Lay down the proof stack

- Mount **Updates/Drivers USB**:
  - Apply cumulative updates (.msu/.cab) in ascending order (Servicing Stack first).
  - Install chipset/storage/GPU drivers via pnputil or Device Manager "Have Disk…".
- Copy your offline kit to D:\proofbox\:
  00_docs\, 01_kits\, 02_toolchain\, 03_inputs\, bin\, keys\, logs\.

Sanity checks (before first proof)

- Open 00_docs\index.html (Help must be local).
- pip install from 02_toolchain\wheels with PIP_NO_INDEX=1.
- cargo build --locked --offline against your vendored sources.

- Verify environment pins: PATH order, PYTHONHASHSEED, temp dir, compiler flags.

First proof run (P0)

- Seed: 12345.
- Run: python run_proofs.py --suite P0 --seed 12345 --out D:\proofbox\04_runs\2025-09-24_2200\outputs
- Produce and sign manifest (detached): make_run_manifest.ps1 (from your bin\).
- Confirm identical hashes on a 2nd run (determinism).

Typical gotchas & fixes

- **Hidden fetch at build:** a post-install script tries to "curl" something. Remove or vendor it; rebuild.
- **Path leaks in binaries:** disable absolute path embedding (/Brepro, -ffile-prefix-map, no __DATE__/__TIME__).
- **Time warnings:** if signed 3rd-party binaries complain about CRL/OCSP, do not introduce them post-trust-epoch; keep only artifacts you already validated.

9.2 Driver + firmware strategy without calling home

Inventory once, then freeze

- Collect PNP hardware IDs → map to driver packages and versions. Save as drivers_inventory.csv.
- Stage drivers under 01_kits\drivers\vendor_model_version\ with vendor checksums and your sha256.

Install/update drivers (offline)

- Prefer pnputil for import/install:
- pnputil /add-driver D:\proofbox\01_kits\drivers**.inf /install
- pnputil /enum-drivers > D:\proofbox\logs\drivers_after.txt
- Verify Device Manager shows **0 unknown devices**. Snapshot that view (screenshot) into logs\.

Firmware (UEFI/NVMe/Wi-Fi if present)

- Prefer **USB-bootable** vendor updaters or capsule updates that run pre-boot.
- Store the exact updater + vendor checksum in 01_kits\firmware\device_version\.
- Update only during maintenance windows; log: start/end, version before/after, sha256 of the updater used.

No portals, no "assistants"

- Avoid vendor utilities that require a live portal (auto-update apps). Use the bare .exe/.bin update with a published checksum.
- If a component's only path is "online tool," treat that component as **non-updatable** in the air-gap and select a different hardware SKU next cycle.

Acceptance drill (quarterly)

- Reinstall one driver from 01_kits using pnputil; reboot; confirm the version matches inventory.
- If you keep one firmware update on hand, apply it once per year during a restore drill and verify the system still builds/proofs identically afterward (ERL-5 test).

9.3 The daily loop: ingest → verify → stage → run → attest

0) Start-of-day prep (2 minutes)

- Set local time per policy; append to time_log.txt.
- Ensure FERRY_A's hardware **RO switch is ON** before you insert it.
- Confirm NICs remain disabled (BIOS + Device Manager).

1) Ingest (from FERRY_A to 90_quarantine)

robocopy F:\2025-09-24_pack D:\proofbox\90_quarantine\2025-09-24_pack /E /R:1 /W:1

copy F:\2025-09-24_pack.zip D:\proofbox\90_quarantine\

copy F:\2025-09-24_pack.zip.minisig D:\proofbox\90_quarantine\

- Append a custody entry (media in/out, RO→RW transitions).

2) Verify (signatures + hashes)

cd D:\proofbox\90_quarantine

D:\proofbox\bin\minisign.exe -Vm 2025-09-24_pack.zip -P D:\proofbox\keys\minisign.pub

D:\proofbox\bin\minisign.exe -Vm .\2025-09-24_pack\MANIFEST_SHA256.txt -P D:\proofbox\keys\minisign.pub

powershell -NoP "$m = Get-Content .\2025-09-24_pack\MANIFEST_SHA256.txt; foreach($line in $m){ $h,$p=$line -split ' ',2; if((Get-FileHash $p -Algorithm SHA256).Hash -ne $h){ throw \"MISMATCH $p\" } }"

- If **any** mismatch: stop, quarantine the pack, note in ingress.log with result=FAIL.

3) Stage (move verified content to 91_stage)

robocopy D:\proofbox\90_quarantine\2025-09-24_pack D:\proofbox\91_stage\2025-09-24_pack /E /COPY:DAT /DCOPY:T

4) Release (to kits/toolchain/inputs)

- Decide the destination once; do not scatter files.
- Record in ingress.log:

ts_local,pack_name,src_media,bytes,files,sha256_zip,signer_pubkey_short,result,released_to

- Move into place (example to wheels):

robocopy D:\proofbox\91_stage\2025-09-24_pack D:\proofbox\02_toolchain\wheels /E /MOV

5) Run (builds and proofs)

- Open a new run directory: D:\proofbox\04_runs\2025-09-24_2200\.
- Set deterministic env:
    - PYTHONHASHSEED=12345
    - SOURCE_DATE_EPOCH=1732406400 (example fixed epoch)
    - TEMP/TMP = D:\proofbox\tmp
- Build:
    - C/C++: same flags as locks; ensure /Brepro or path-stripping flags set.

    - Python: pip install --no-index --find-links ... -r locks\requirements.txt
    - Rust: cargo build --locked --offline
- Proof:
- python run_proofs.py --suite P0 --seed 12345 --out .\outputs

6) Attest (manifest + signatures)

D:\proofbox\bin\make_run_manifest.ps1 -RunDir D:\proofbox\04_runs\2025-09-24_2200

- This writes manifest.sha256. Sign it **outside** if you keep the private key off-box, or sign with a designated offline signer machine:

minisign -S -s X:\private\minisign.key -m D:\proofbox\04_runs\2025-09-24_2200\manifest.sha256

- Append to egress.log if you export any artifacts to OFFBOX_R:

ts_local,run_id,artifact,bytes,sha256,sig,media,notes

7) Bench snapshot (optional daily)

- Run bench.ps1 -Name proof_P0 -N 3 to spot drift. Target cv_pct <= 5.
- Append to bench_history.csv.

8) End-of-day

- Rotate FERRY_A to **RO**; remove and store.
- Append quick summary to D:\proofbox\logs\daily.txt:
    - time set, packs ingested (OK/FAIL), runs executed, failures observed, next actions.

Failure patterns you'll actually see

- **"pip tries to reach pypi"**: missing PIP_NO_INDEX=1 or a stray dependency without a wheel in your mirror.
- **"cargo wants crates-io"**: forgot replace-with = "vendored-sources" or --locked --offline.
- **"Build hashes differ"**: path/time embedded; audit compile flags and temp dir, check banner macros.
- **"Run time jitter high"**: power plan reverted, thermal throttling, background tasks re-enabled.

What “good” looks like (daily)

- 0 outbound attempts (firewall/ETW logs quiet).
- Identical manifest.sha256 for repeated runs on same inputs.
- Bench cv_pct steady; small drift explained in daily.txt.
- Logs (custody/ingress/egress) updated and human-readable.

That’s your Beelink case study end-to-end: from the first offline boot to a signed, reproducible proof run—plus the muscle memory you’ll actually use every day.

10) What this changes in you

10.1 Attention without pings: flow as a system feature

Quiet is not absence; it’s bandwidth you get back.
An exoplanet-ready box has no inbox to defend. With no latent socket to the world, you stop budgeting cognition for interruptions you “might” get. Three practical shifts show up fast:

- **Longer half-life of focus.** Context survives across minutes and hours because nothing external competes for it. You notice micro-rhythms (compile→inspect→tweak) and tune them, rather than fighting to resume them.
- **Tighter feedback loops.** When you can predict that a build takes ~8.4s (not 6–22s), you plan micro-experiments differently: smaller deltas, more runs, faster learning.
- **Intentional ingress.** Every new dependency must cross your shuttle SOP. Decisions to import are conscious; so is the awareness of cost. Intake becomes curation, not grazing.

Practice: Start sessions with a written “three-move plan” and a stop rule. End sessions by appending a 5-line “where I leave the rope” note for tomorrow. Your future self now trusts the environment and spends near-zero time reorienting.

10.2 Sovereign failure modes (and why they’re kinder)

Offline shifts how things break.
Networked systems fail in tangled ways: expired tokens, partial rollouts, registry timeouts, version drift, unseen policy flips. Air-gapped systems still fail—but *locally* and *legibly*:

- **Failures are bounded.** When something breaks, the blast radius is the working directory and today’s run, not a global outage or a disappearing dependency upstream.

- **RCA is tractable.** You can bisect the last 10 changes because those are the only 10. Inputs are stable, versions pinned, logs complete. Root causes emerge in hours, not weeks.
- **Reversibility is normal.** You can always roll back to the golden image you control. No external service has to say "OK".
- **Moral injury declines.** You stop carrying the latent anxiety that a distant service will change rules mid-work. The machine's promise is humble but kept: it will do what it did yesterday.

Practice: Treat "breaks" as data. Keep a small ledger: trigger, symptom, minimal fix, and whether the fix belongs in locks, SOP, or image. Over time you evolve from firefighting to gardening.

10.3 Choosing dependence deliberately

Dependence isn't evil; unexamined dependence is.
The point is not to live forever offline; it's to make reconnection a *choice* with known tradeoffs.

- **Three buckets for every feature F:**
    1. **Essential:** gates boot/install/build/prove/restore. Must be satisfiable offline.
    2. **Comfort:** makes life nicer (autocomplete, cloud search). Never a prerequisite.
    3. **Negotiable risk:** e.g., a one-time online lookup you'll mirror and then sever.
- **Reconnection rituals:** If you decide to connect (even briefly, on a different machine), do it with a script: fetch → hash → sign → shuttle. Bring back artifacts, not entanglements.
- **Exit tests before entry.** Any tool you add must pass: (a) local docs exist, (b) no forced sign-in for core use, (c) offline install path documented, (d) deterministic builds preserved.
- **Periodic "subtraction days."** Once a quarter, remove one thing you haven't *really* needed. The stack gets lighter; you get braver.

Practice: Write a one-page "Dependence Charter" and tape it to the chassis. It lists your essentials, your allowed comforts, and the tests any new tool must pass. When a shiny thing beckons, you have a standard higher than impulse.

Net effect: Attention consolidates, failures get kinder, and dependence becomes contractual, not habitual. You don't become a hermit; you become a careful citizen of two worlds—one quiet enough to think in, and one loud enough to trade with—crossing by ferry, on your terms.

11) Policy and pedagogy

11.1 "Offline-first" as a right, not a curiosity

Principle statement (plain language):
Users must be able to install, operate, repair, and restore general-purpose computers **without** a network connection. Offline operation is not an edge case; it is a safety, research, and civil-liberties requirement.

Why it belongs in policy

- **Safety & resilience:** Critical work (research, medical devices, elections, field ops) must survive network loss.
- **Reproducibility:** Science and engineering require stable environments and audit trails.
- **Privacy by construction:** No telemetry is the simplest privacy guarantee.
- **Equity:** Not all learners or sites have reliable connectivity.

Model policy clauses (drop-in text)

- **Right to local account:** "All organization-issued computers SHALL support creation of a local account during initial setup without requiring internet access or a cloud identity."
- **Right to offline install:** "All essential software SHALL provide an offline installer that completes without contacting external services."
- **Right to static docs:** "Vendors SHALL provide static documentation (PDF/EPUB/CHM) for the shipped version, licensed for local distribution."
- **Ban on coercion UX:** "Setup flows MUST present offline paths with equal prominence; dark-pattern gating to force sign-in or connectivity is prohibited."
- **Testable assurance:** "Procurement teams SHALL validate offline install and operation prior to acceptance."

Governance & audits

- Add “Offline Readiness” to security and accessibility reviews.
- Track an **ERL score** per fleet (target ERL-2 for general use; ERL-4+ for labs/research).

11.2 Vendor minimums: local account, local docs, local install paths

Minimums checklist (procurement one-pager)

1. **Local account path** documented and stable across releases.
2. **Offline installers** (OS, SDKs, drivers) with vendor SHA256 checksums.
3. **Static docs** matching shipped versions (not “latest only”).
4. **Deterministic builds** possible without hidden network fetches (no forced template/download steps).
5. **Config flags** to disable telemetry and network lookups.
6. **Driver/firmware packages** available as standalone downloads (no portal lock-in).
7. **License terms** that permit offline activation or perpetual offline use after initial activation.

Simple vendor attestation (RFP appendix)

- “Our product can be installed and fully operated offline.” YES/NO
- “A local account path exists in OOBE/first-run.” YES/NO
- “We supply static PDFs/EPUBs for the shipped version.” YES/NO
- “We provide SHA256 checksums for installers and updates.” YES/NO
- “We support disabling telemetry via documented flags.” YES/NO
- “We publish driver/firmware as standalone packages.” YES/NO
- If any NO: attach remediation plan and timeline.

Acceptance test (30 minutes per product)

- Boot a clean machine with radios disabled.
- Install the product from removable media only.
- Open local docs (no URLs).

- Perform a representative build/run.
- Confirm zero outbound attempts (firewall log stays quiet).
  If any step fails, mark **provisionally non-compliant** and request fixes.

11.3 Teaching computing that still works on an exoplanet

Course framing (1–2 weeks micro-module or a full term)

- Title: *Offline-First Computing: Reproducible Systems without the Web*
- Learning outcomes:
  - Students can install and operate a dev environment **fully offline**.
  - Students can produce **reproducible builds** and **attestable artifacts**.
  - Students understand and apply **Exoplanet Readiness Levels (ERL)**.

Syllabus outline

- **Week 1: Concepts & threat model**
  - Lecture: Dependence vs sovereignty; air-gap benefits/limits.
  - Lab: Disable radios, create local accounts, set up offline time policy.
- **Week 2: Packaging an offline universe**
  - Lecture: Mirrors, lockfiles, static docs.
  - Lab: Build 00_docs/ and 02_toolchain/ from provided media.
- **Week 3: Determinism**
  - Lecture: Sources of nondeterminism (time, paths, locales).
  - Lab: Produce bit-identical builds; fix a path/time embedding bug.
- **Week 4: Shuttle SOP & attest**
  - Lecture: Hashes, signatures, custody logs.
  - Lab: Ingest → verify → stage → run → manifest → sign → export.
- **Week 5: ERL drills & recovery**
  - Lecture: ERL-0..5; bare-metal imaging.

- Lab: Wipe → restore from image → verify hashes → run benchmark.

Lab kits (fits in a shoebox)

- 2× USB sticks (FERRY_A with RO switch, OFFBOX_R).
- Printed **quick-start** with SOPs and ERL checklists.
- Pre-curated mirrors (Python wheels, cargo vendor, .NET layout) on instructor media.

Rubrics (pass/fail gates)

- **ERL-1 check:** Cold install & patch from USB; Device Manager clean; docs open offline.
- **ERL-2 check:** Build a small project using only file:// mirrors; no outbound attempts.
- **ERL-3 check:** Two rebuilds yield identical manifest.sha256; bench cv_pct ≤ 5.
- **ERL-4 check:** Bare-metal restore succeeds; hashes match canonical manifest.

Assessments

- **Practical exam (90 min):** Given a bare machine and a kit, reach ERL-2 and demonstrate a deterministic build.
- **Reflection (1 page):** "What this changed in my debugging practice."
- **Team drill:** Triage a broken build *without* the internet (pair bisection with logs).

Inclusion & accessibility

- All required materials (installers, docs, datasets) supplied on physical media.
- No telemetry or cloud accounts required.
- Provide PDF/EPUB docs for screen readers; allow keyboard-only workflows.

Capstone ideas

- **Portable proofbox:** Students assemble a mini-PC that reaches ERL-4 and demo a timed restore.
- **Offline reproducible paper:** Re-implement a small algorithm and ship a signed artifact pack with a rebuild script and manifest.
- **Policy memo:** Students audit a popular SDK against the vendor minimums and draft a compliance plan.

Program outcomes

- Graduates can *prove* their work: not only that it runs, but that it will still run in a year, on a ship, or—yes—on an exoplanet.

If you'd like, I can turn the policy clauses into a one-page "Offline-First Procurement Addendum," plus a 5-lab instructor packet (slides + handouts + grading sheets) aligned to ERL checkpoints.

12) Conclusion — The quiet machine and the loud world: where to reconnect, and where not to

The point of the exercise wasn't to swear off the network. It was to make the network optional. Once you've felt the stability of an exoplanet-ready box—predictable builds, identical proofs, legible failures—you see the online world differently: as a marketplace you visit on purpose, not a pipeline that owns your runtime.

Where to reconnect (deliberately)

- **Discovery & research (off-box):** Find ideas, papers, and packages **on a separate, online machine**. Curate, then shuttle artifacts (hashed/signed) across.
- **Security intake (curated):** Periodic offline updates to your kit—OS cumulative updates, compiler point releases, firmware you've validated. Treat this like changing the oil: scheduled, logged.
- **Publishing & collaboration (egress-only):** Export signed artifacts and manifests for others. Share build instructions, not your build environment.
- **One-time bootstraps:** Initial license activations or key exchanges—done once, recorded, then severed.

Where *not* to reconnect (ever)

- **OOBE/identity gates:** No cloud logins for setup or core use.
- **Toolchain operation:** No live registries, "fetch at build," or template downloads.
- **Help & docs:** Never depend on web links during debugging or class. Local shelf or it doesn't count.

- **Determinism breakers:** Anything that injects time, path, or remote state into builds/proofs.
- **Background daemons:** Updaters, telemetry, “assistants” that create jitter and leak context.

How to keep the boundary healthy

- **Run ERL drills** quarterly (install, build, restore). Initial next to each checklist.
- **Rotate the shuttle** (media replacement, hash your logs, sign the latest manifests).
- **Pin, then review:** Locks freeze today; a scheduled review decides what to thaw.
- **Tiny change budget:** Prefer small, testable deltas to “big bang” upgrades.
- **Paper trail > memory:** Custody, ingress, egress, bench history—boring on purpose.

A simple decision tree

1. Does this feature gate boot/install/build/prove/restore?
   Yes → Must work offline (essential) or be replaced.
   No → Comfort; keep disabled by default.
2. If connected, can the same result be achieved by **fetch → hash → sign → shuttle**?
   Yes → Reconnect off-box only; import artifacts, not dependencies.
   No → Reconsider the tool or redesign the workflow.

Final stance: Keep the machine quiet and the choices loud. Reconnect where it creates durable value you can bring back as signed, local artifacts. Refuse connection where it steals determinism, privacy, or time. That’s not isolationism; it’s craft.

Sidebar A — Air-gap Ops Playbook (10 rules)

1) Two USBs: ferry and quarantine—never reuse across roles.

Why: Role separation prevents cross-contamination.
Do:

- Label **FERRY_A** (exFAT, has HW read-only switch) → moves between online/offline.
- Label **QUAR_Q** (NTFS) → **lives only** on the offline box as first landing zone.
  Ops tip: Flip FERRY_A to **RO** before entering the offline room. Never plug FERRY_A into a third machine.

---

2) Hash everything: sha256sum > LOG; verify before mount/use.

Why: You can't "eyeball" integrity.
Do (online):

```
# Windows (PowerShell)

Get-ChildItem -Recurse | Get-FileHash -Algorithm SHA256 |

 ForEach-Object { "{0}  {1}" -f $_.Hash, $_.Path } |

 Set-Content MANIFEST_SHA256.txt
```

Do (offline): Recompute and compare; **no promote** until every line matches.

---

3) Sign releases: detached signatures with a long-lived offline key.

Why: Authenticates source, not just bits.
Do: Keep private key on a **never-shuttled** USB. Use minisign or GPG.

```
# Sign on signing box

minisign -S -s X:\keys\minisign.key -m MANIFEST_SHA256.txt

# Verify offline box

minisign -Vm MANIFEST_SHA256.txt -P D:\proofbox\keys\minisign.pub
```

Practice: Rotate keys on a calendar, not in a crisis. Print the public fingerprint.

---

4) No autorun, ever; open via hash-listed paths only.

Why: Autorun is an execution trap.
Do (Windows):

reg add HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Policies\Explorer /v NoDriveTypeAutoRun /t REG_DWORD /d 255 /f

reg add HKCU\Software\Microsoft\Windows\CurrentVersion\Policies\Explorer /v NoDriveTypeAutoRun /t REG_DWORD /d 255 /f

Discipline: Never double-click files in 90_quarantine\. Promote → release → open from curated dirs only.

---

5) Convert docs to stable, offline formats (PDF/CHM/EPUB).

Why: Links rot; formats drift. PDFs don't.
Do (online): Save vendor manuals as **PDF/EPUB**, not web bookmarks.
Policy: Reject macro-enabled Office docs; if needed, convert upstream to PDF/A before shuttling.
Shelving: Keep 00_docs\index.html as your table of contents with local links.

---

6) Mirror once, use for months: versioned snapshots beat "latest".

Why: "Latest" breaks reproducibility.
Do: Snapshot **wheels**, **cargo vendor**, **pnpm store**, **.NET layout**, **C/C++ libs** under 02_toolchain\ with **locks**.
Rhythm: Refresh on a schedule (e.g., quarterly), not ad-hoc. Treat updates like releases: test → bless → snapshot.

---

7) Log ingress/egress in plain text with timestamps.

Why: A boring flight recorder solves real mysteries.
Do: Three CSVs: custody.log, ingress.log, egress.log. Append, never overwrite.
Example (ingress):

2025-09-24T20:30,2025-09-24_pack,FERRY_A,18344219,142,ABCD...9912,minisign,OK,02_toolchain\wheels

Habit: Print the last page weekly and initial it.

8) Prefer static builds; avoid dynamic plugins.

Why: DLL/plugin discovery adds nondeterminism and attack surface.
Do:

- Link statically when feasible (/MT on MSVC, static CRT/libs).
- Disable runtime downloads, autoloaded plugins, “fetch at build”.
- Vendor source or exact binaries; lock versions.
  Test: Repeat builds 3× → identical hashes or you’re not done.

---

9) Keep a golden bare-metal image and test restores quarterly.

Why: Backups you’ve never restored are fiction.
Do (Windows/WinPE):

Capture

dism /capture-image /imagefile:D:\images\golden_os.wim /capturedir:C:\ /name:"golden_os" /compress:max /checkintegrity

Restore drill

dism /apply-image /imagefile:D:\images\golden_os.wim /index:1 /applydir:C:\

bcdboot C:\Windows /s S:\EFI /f UEFI

After restore: Verify hashes of D:\proofbox\ against canonical manifest; run a micro build.

---

10) Treat time as data: record local time policy in every run log.

Why: Time without NTP must be explicit.
Do: Disable W32Time, set time **once per session**, and log:

2025-09-24T19:58 local | set by: Doug | ref: Seiko | delta: +00:02:15

Builds: Use SOURCE_DATE_EPOCH for reproducible timestamps; use monotonic clocks for benchmarks; never backdate after signing.

60-second daily checklist

- FERRY_A switch **RO** before insert.
- Ingest → **verify sigs/hashes** → stage → release.
- Build/prove with pinned env; emit manifest.sha256.
- Update **ingress/egress/custody** logs.
- Jot time entry + 3-line “where I leave the rope.”

If you want, I can bundle these into a printable A4 card and a one-click PowerShell script that enforces items #2, #4, #7, and #10 on login.

Sidebar B — OOBE & vendor lock-ins (high-level, ethical)

Green paths (do these)

- **Use documented offline paths.** Favor SKUs/editions and setup modes that explicitly support a **local account** and **offline provisioning** (answer files, deployment images, preseed/kickstart, MDT/Autopilot-offline, etc.).
- **Prebuild, don't wrestle.** Create your own **reference image** with a local admin, policies, drivers, and updates baked in; deploy from removable media. Avoid clicking through surprise wizards.
- **Prefer standalone packages.** OS updates, drivers, SDKs, and firmware should be available as **offline installers with vendor checksums**. Keep those plus your own sha256 in the kit.
- **Disable net-assumptions up front.** In firmware: turn off LAN/Wi-Fi/BT and PXE. In OS: block "search online for drivers," disable background sync/telemetry via documented settings.
- **Keep docs local.** Ship static PDFs/EPUBs for the versions you actually install; wire "Help" to file:// locations.

Red flags (treat as blockers)

- "**Sign in to continue**" with no first-class local path.
- "**We'll finish setup after you connect**."
- Installers that are **web stubs** (download-at-runtime).
- Tools that **revalidate licenses periodically** without an offline option.
- Firmware delivered **only** through online portals/assistants (no standalone capsule/updater).
  When any of these appear, stop and switch to a compliant method (imaging/provisioning) or a different product.

Avoid gray-area hacks

- Don't depend on registry pokes, MITM tricks, or EULA-dodging to "skip" OOBE gates. They're brittle, often violate terms, and can silently disable security features. If the **official** offline path doesn't exist or regresses, treat it as a **vendor issue**, not a puzzle to beat.

Deployment tools (ethical menu)

- **Answer files / unattended setup** (documented keys only).
- **Sysprep + capture / WinPE imaging** (or your platform's supported equivalent).
- **Offline package "layout"** features (SDKs, VS/.NET, NuGet local sources).
- **Driver import** via signed .inf + pnputil (or platform's offline driver tooling).
- **Firmware** via vendor-published standalone updaters with checksums.

Your paper trail (keep this)

- **OOBE transcript:** Every screen you accepted/declined; exact build/edition.
- **Network exceptions log:** Any step that *could* use the net (activation, driver search) and why you declined or how you satisfied it offline.
- **Image recipe:** Hashes of installers/updates, driver versions + PNP IDs, policies applied, Secure Boot/TPM state.
- **Variance report:** If a later build alters wording/flow, capture screenshots and note the compliant alternative you used (e.g., deployed image instead of interactive OOBE).

Templates (copy/paste)

OOBE Transcript (excerpt)

Device: Beelink R7 | Firmware: v1.12 | Date: 2025-09-24 (local)

Screen 1: Region/KB -> US/US (offline)

Screen 2: Connect to network -> "I don't have Internet" (offline path chosen)

Screen 3: Account -> "Offline account" -> Local admin 'cluster'

Screen 4: Privacy/telemetry -> All toggles 'Off' where offered

Post-OOBE: Radios confirmed disabled (BIOS + OS), Outbound firewall = Block

Network Exceptions Log (excerpt)

2025-09-24 20:05 Activation -> Deferred; SKU supports offline activation; see binder p.12

2025-09-24 20:18 Drivers -> Installed from 01_kits\drivers\ (no online search)

2025-09-24 20:40 SDK setup -> Used offline layout at 02_toolchain\dotnet\layout

Imaging Recipe (excerpt)

Base ISO: Win 11 Pro 24H2, SHA256=...

Updates: KB50xxxx (SSU first), SHA256=...

Drivers: AMD Chipset 5.x, NVMe 2.x, SHA256 list stored in drivers_manifest.txt

Policies: Disable NTP, block device metadata retrieval, disable Store auto-installs

Secure Boot: Enabled (stock keys) | TPM: Enabled | BitLocker: Off (air-gap)

Procurement one-pager (hand this to vendors)

- **Local account path at setup** (documented, stable).
- **Offline installers** for core product + workloads (checksums published).
- **Static docs** for the shipped version (PDF/EPUB).
- **No forced periodic online validation** for offline use.
- **Standalone drivers/firmware** (no portal-only delivery).
- **Configurable telemetry (opt-out)** via documented flags.
  Require a **Yes/No** attestation for each; test on a radio-disabled machine before acceptance.

Ethics filter (3 questions)

1. **Legitimacy:** Is the offline path supported in docs/EULA?
2. **Stability:** Will this method survive an update (image > UI trick)?
3. **Safety:** Does this preserve security features (Secure Boot/driver signing) rather than disabling them?

If any answer trends “no,” change the *method* or the *vendor*, not your standards.

Bottom line: Choose the **supported** offline roads, document them, and hold the line. You’re not being difficult; you’re making computing that still works when the network doesn’t—and that’s an ethical requirement, not a novelty.

Appendix — Minimal offline kit (practical checklist)

0) Labeling & media (non-negotiable)

- **USB-A #1 — INSTALL** (32–64 GB, fast, RW)
  OS installer + offline updates + WinPE/Recovery.
- **USB-A #2 — KITS** (128–256 GB, RW)
  Toolchains, mirrors, drivers, firmware, docs.
- **USB-A #3 — FERRY_A** (64–128 GB, **hardware RO switch**)
  Shuttle between online/offline (switch **RO** before entering the lab).
- **USB-A #4 — OFFBOX_R** (64–128 GB, RW)
  Export signed artifacts out of the air-gap.
- **USB-A #5 — SIGNING** (8–32 GB, stays locked away; **never** shuttled)
  Private keys only.
- **A4 binder** — printed runbooks, fingerprints, ERL checklists.

Tip: big, ugly labels: INSTALL, KITS, FERRY_A, OFFBOX_R, SIGNING. Date them.

---

1) BIOS/UEFI update on a throwaway USB (verify offline)

Why: Modern firmware updaters can run from USB/UEFI shell with vendor checksums.

What to pack on KITS\01_kits\firmware

- vendor_readme.txt (copied from vendor page).
- uefi_firmware.bin (or .cap/.exe), versioned folder: beelink_r7_v1.12\.
- vendor_sha256.txt (official) + **your** sha256sums.txt.

Verify (online box, once)

Get-FileHash .\uefi_firmware.bin -Algorithm SHA256 > sha256sums.txt

Print both vendor and local hashes for the binder.

Run (offline, maintenance window)

- Boot to vendor firmware utility or UEFI shell.
- Apply update from INSTALL or a **throwaway** USB.

- Record before/after version + file hash in D:\proofbox\logs\firmware_log.csv.

---

2) OS install media + cumulative updates on USB; scripted post-install

INSTALL USB contents

\OS\Win11_24H2.iso

\WinPE\bootable_recovery_usb\

\Updates\SSU_KBxxxxx.msu

\Updates\LCU_KBxxxxx.msu

\Drivers\Base\ (storage/nvme/chipset if needed for setup)

\PostInstall\post_install.ps1   (your hardening script)

\Hashes\install_media.sha256

Post-install hardening script (example lines)

```
# post_install.ps1 (run as admin)

# Disable NTP, set outbound firewall default to Block, stop consumer features.

Stop-Service W32Time; Set-Service W32Time -StartupType Disabled

New-NetFirewallProfile -Profile Domain,Public,Private -DefaultOutboundAction Block -DefaultInboundAction Block

reg add "HKLM\SOFTWARE\Microsoft\Windows\CurrentVersion\Device Metadata" /v PreventDeviceMetadataFromNetwork /t REG_DWORD /d 1 /f

# Create dirs

New-Item -ItemType Directory D:\proofbox\{00_docs,01_kits,02_toolchain,03_inputs,04_runs,90_quarantine,91_stage,bin,keys,logs} | Out-Null
```

Apply updates (offline)

```
wusa.exe .\Updates\SSU_KBxxxxx.msu /quiet /norestart

wusa.exe .\Updates\LCU_KBxxxxx.msu /quiet /norestart
```

---

3) Language toolchains, compilers, package mirrors (pinned)

KITS\02_toolchain\ layout

wheels\ # Python wheels

rust\vendor\ # cargo vendor snapshot

node\store\ # pnpm store or npm tarballs

dotnet\layout\ # VS/.NET offline layout

cxx\libs\ # static libs + headers

locks\ # requirements.txt, Cargo.lock, pnpm-lock.yaml, *.lock

manifests\toolchain_sha256.txt (and .minisig)

Collect on the online box

- Python:
- pip download -r requirements.txt -d .\wheels
- python -V > .\locks\python.lock
- pip --version >> .\locks\python.lock
- pip freeze -r requirements.txt >> .\locks\python.lock
- Rust:
- rustup toolchain install 1.80.1
- cargo vendor --versioned-dirs --locked --respect-source-config --sync .\Cargo.lock .\rust\vendor
- rustc --version > .\locks\rust.lock
- cargo --version >> .\locks\rust.lock
- Node (pnpm):
- pnpm install
- pnpm store path # copy path contents into node\store\
- .NET / VS:
- vs_buildtools.exe --layout C:\kit\dotnet\layout --lang en-US

- dotnet-sdk-*.exe (offline installers)

Pin managers to file:// (offline box)

- Pip: set PIP_NO_INDEX=1, PIP_FIND_LINKS=file:///D:/proofbox/02_toolchain/wheels
- Cargo: .cargo/config.toml → offline = true, replace-with = "vendored-sources"
- pnpm: .npmrc → offline = true, store-dir = D:\proofbox\02_toolchain\node\store
- NuGet: local source only in NuGet.Config

---

4) Local doc set (language refs, manpages, PDFs, API docs)

KITS\00_docs

index.html

specs\python_3.12.5_ref.pdf

manuals\windows_24H2_admin.pdf

apidocs\rust\ (from `cargo doc --offline`)

apidocs\project\ (Sphinx/Doxygen output)

manifest\docs_map.txt

manifest\docs_sha256.txt (+ .minisig)

Quick index builder (PowerShell)

```
$root="D:\proofbox\00_docs"
Get-ChildItem $root -Recurse -Include *.pdf,*.html,*.epub |
  Get-FileHash -Algorithm SHA256 |
  ForEach-Object { "{0}  {1}" -f $_.Hash, $_.Path } |
  Set-Content "$root\manifest\docs_sha256.txt"
```

---

5) Hash/sign tools + printed fingerprint

On KITS\bin

- minisign.exe, sigcheck64.exe, sha256sum.exe (or PowerShell Get-FileHash).
- verify_and_ingress.ps1, make_run_manifest.ps1, bench.ps1 (from earlier sections).

Key handling

- Generate minisign keypair on a **separate, never-networked** laptop:
- minisign -G -p public.pub -s private.key
- Put public.pub on the offline box (D:\proofbox\keys\minisign.pub).
- Keep private.key on **SIGNING** USB (safe).
- **Print** the public key fingerprint for the binder.

---

6) A4 binder: runbooks, recovery, ERL rubric (printed)

Print and hole-punch

- **Front pocket:** Public key fingerprint(s), emergency contacts.
- **Tabs:**
    1. **ERL Checklists** (ERL-0…ERL-5 one-pagers)
    2. **Imaging & Restore** (WinPE steps, DISM commands, hash verifies)
    3. **USB Shuttle SOP** (ingest → verify → stage → release)
    4. **Build/Proof SOP** (env vars, flags, seed policy)
    5. **Time Policy** (how/when set, monotonic notes)
    6. **Manifests** (last signed toolchain_sha256.txt, docs_sha256.txt)
    7. **Logs (printouts)** last pages of custody/ingress/egress
    8. **Drivers/Firmware Matrix** (PNP IDs → versions, hashes)

---

7) Packing list (ready-to-go file tree on KITS)

\00_docs\...

\01_kits\

  \drivers\...

  \firmware\...

  \updates\...

\02_toolchain\

  \wheels\ \rust\vendor\ \node\store\ \dotnet\layout\ \cxx\libs\

  \locks\ \manifests\toolchain_sha256.txt toolchain_sha256.txt.minisig

\03_inputs\examples\...

\bin\minisign.exe \bin\sigcheck64.exe \bin\verify_and_ingress.ps1
\bin\make_run_manifest.ps1 \bin\bench.ps1

\keys\minisign.pub

\logs\ (empty; created on box)

---

8) First-hour drill (proves the kit works)

1. **BIOS radios off.** Boot from INSTALL.
2. **Install OS**, run post_install.ps1.
3. **Apply updates** from INSTALL\Updates.
4. **Copy KITS** to D:\proofbox\.
5. **Open docs** via 00_docs\index.html.
6. **Install Python deps** from wheels; **cargo build** --offline; **pnpm install** from local store.
7. **Run toy proof**, produce manifest.sha256, and (optionally) sign on the signer.
8. **Log** entries: custody/ingress/egress; print the summary sheet for the binder.

Pass if: no network prompts, all installs/builds succeed, docs open locally, hashes/signatures verify.

---

9) Maintenance cadence (write this on the binder cover)

- **Quarterly:** ERL-4 restore drill + bench snapshot; print last page of logs.
- **Quarterly:** Refresh KITS mirrors (optional), re-sign new toolchain_sha256.txt.
- **Semiannual:** Firmware/driver review (only from standalone updaters).
- **Annual:** Rotate FERRY_A media; re-print fingerprints and check runbooks match reality.

---

10) One-page mini-checklist (tear-out)

- Radios off (BIOS & OS)
- Local admin created; NTP disabled; time set & logged
- INSTALL USB boots; updates apply offline
- KITS copied; hashes verified; signatures OK
- Package managers pinned to file:// mirrors
- Docs shelf loads from 00_docs\index.html
- Deterministic env vars set (PYTHONHASHSEED, SOURCE_DATE_EPOCH)
- Build & proof run complete; manifest.sha256 written (+ signed)
- Logs updated (custody/ingress/egress); binder page printed

This is the smallest kit that still lets you **install, build, prove, attest, and restore** without a network—and prove it on paper.

References

Standards & Core Specs

1. NIST. *FIPS 180-4: Secure Hash Standard (SHA-256, -384, -512).*
2. IETF. *RFC 5905: Network Time Protocol Version 4 (NTPv4).*
3. UEFI Forum. *Unified Extensible Firmware Interface (UEFI) Specification.*
4. NIST. *SP 800-53 Rev. 5: Security and Privacy Controls for Information Systems and Organizations.*
5. Reproducible Builds Project. *SOURCE_DATE_EPOCH Specification & Best Practices.*

Cryptographic Signing & Integrity

6. Frank Denis. *Minisign & Ed25519: Simple File Signing.*
7. The GNU Privacy Guard. *GnuPG Handbook.*
8. Microsoft Sysinternals. *Sigcheck: File Version and Signature Viewer.*

Imaging, Installation & Recovery (Windows)

9. Microsoft. *DISM (Deployment Image Servicing and Management) Technical Reference.*
10. Microsoft. *WinPE (Windows Preinstallation Environment) User's Guide.*
11. Microsoft. *BCDBoot Command-Line Options.*
12. Microsoft. *WUSA: Windows Update Standalone Installer.*
13. Microsoft. *PnPUtil Command-Line Options (Driver Package Management).*

Windows Configuration, Policy & Networking

14. Microsoft. *Windows Time Service (W32Time) Architecture and Management.*
15. Microsoft. *Windows Defender Firewall with Advanced Security—Outbound/Inbound Defaults.*
16. Microsoft. *Disable AutoRun and AutoPlay (Group Policy / Registry).*
17. Microsoft. *Device Installation Settings & "Search Windows Update for Drivers."*
18. Microsoft. *Power Plans & powercfg Documentation.*
19. Microsoft. *Event Tracing for Windows (ETW) Overview.*

Package Managers & Offline Mirrors

20. Python Packaging Authority. *pip User Guide (—no-index, —find-links, wheel caching).*
21. Rust. *cargo Book: cargo vendor, --locked, [source] replacement & offline builds.*
22. npm & Node.js. *npm CLI Docs: npm pack, local registries, audits (disable).*
23. pnpm. *pnpm Store & Offline Mode Configuration.*
24. NuGet. *NuGet.Config Package Sources & Global Packages Folder.*
25. .NET & Visual Studio. *Offline "Layout" Installers for VS/Build Tools & .NET SDK.*

Build Systems, Toolchains & Determinism

26. LLVM/Clang & MSVC. *Reproducible Build Flags (e.g., /Brepro, path & timestamp controls).*
27. CMake. *FetchContent/ExternalProject Policies & Offline Configuration.*
28. Doxygen. *Reference Manual (Local HTML Output).*
29. Sphinx. *Documentation Builder (HTML/PDF for Local Shelves).*
30. Rustdoc. *cargo doc Offline Generation.*

Documentation & Local Help

31. Adobe. *PDF/A (ISO 19005) Overview for Long-term Preservation.*
32. CHM/Compiled HTML Help. *Windows Help Formats (local, offline).*

Time, Logging & Benchmarking

33. Microsoft .NET. *System.Diagnostics.Stopwatch (Monotonic Clock Source).*
34. Python. *time.perf_counter() Reference (High-resolution monotonic).*
35. NIST. *Time and Frequency Services (disciplining without NTP—conceptual primers).*

Firmware, Drivers & Hardware

36. PCI-SIG / USB-IF. *Device & Interface IDs (for inventory mapping).*
37. NVMe Express. *NVMe Specifications (controller/firmware update concepts).*
38. AMD. *AMD Chipset & Graphics Driver Packaging (Standalone Installers).*
39. Vendor PC OEM (Beelink). *Model-specific UEFI/Firmware Update Guides & Checksums.*

Air-Gap Operations & Governance

40. NCSC (UK) / CISA (US). *Guidance on Offline/Isolated ("Air-Gapped") Systems.*
41. NIST. *SP 800-171/172 (as applicable) — Controlled Unclassified Information & Enhanced Protections* (offline operation implications).

How to use this list

- **Specs first:** cite #1–5 when justifying hashing, time policy, and reproducible builds.
- **Primary docs:** use #9–19, #20–25 for exact command syntaxes and offline flags.
- **Determinism:** point to #26–30 when explaining why you set timestamp/path controls.
- **Ops & policy:** reference #40–41 when writing procurement or governance language.